NASA/CP—2002–210000

Tenth Goddard Conference on Mass Storage Systems and Technologies

in cooperation with the

Nineteenth IEEE Symposium on Mass Storage Systems

Edited by
Benjamin Kobler, Goddard Space Flight Center, Greenbelt, Maryland
P C Hariharan, Systems Engineering and Security, Inc., Greenbelt, Maryland

Proceedings of a conference held at
The Inn and Conference Center
University of Maryland, University College
College Park, Maryland, USA
April 15–18, 2002

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

April 2002

The NASA STI Program Office … in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.
- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.
- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.
- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and mission, often concerned with subjects having substantial public interest.
- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at http://www.sti.nasa.gov/STI-homepage.html
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA Access Help Desk at (301) 621-0134
- Telephone the NASA Access Help Desk at (301) 621-0390
- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320

NASA/CP—2002–210000

Tenth Goddard Conference on Mass Storage Systems and Technologies

in cooperation with the

Nineteenth IEEE Symposium on Mass Storage Systems

Edited by
Benjamin Kobler, Goddard Space Flight Center, Greenbelt, Maryland
P C Hariharan, Systems Engineering and Security, Inc., Greenbelt, Maryland

Proceedings of a conference held at
The Inn and Conference Center
University of Maryland, University College
College Park, Maryland, USA
April 15–18, 2002

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

April 2002

Available from:

NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076-1320
Price Code: A17

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Price Code: A10

Preface

This volume collects together 27 papers from the Tenth Goddard Conference on Mass Storage Systems and Technologies being held in cooperation with the Nineteenth IEEE Symposium on Mass Storage Systems and Technologies.

The Conference opens on the first day with tutorials on perpendicular recording in magnetic media, IP storage, object-based storage, and storage virtualization. Over the following three days, there are twelve sessions on various themes: Networked Storage, Hierarchical Storage Management, Storage Indexing. Instead of a poster session, the Program Committee decided this year to have a set of shorter papers in the plenary sessions. Time has been set aside for extemporaneous presentations to provide an opportunity for those with a message who either did not write up a paper, or decided, after looking at the program, that they had worthwhile ideas to share.

An invited panel on the third day will cast a look at the future of storage, and reflect also on the past. Intense competition in the disk drive industry has led to mergers and a reduction in the number of manufacturers. The industry, however, has managed to maintain a rate of doubling the areal density every year at least over the last two years. Nanomagnetism and perpendicular recording are two ways to push back the superparamagnetic limit. The tape industry has not achieved the same areal density as their brethren in the disk industry, but a cartridge holding a terabyte of data is now more than just a possibility.

Networked storage (NAS, SAN) is now more prevalent in data centers, and WAN based IP storage has been demonstrated. An interoperability demonstration among different products from various vendors is planned as part of the vendor expo.

Vendor exhibits will continue through the three days of the general sessions.

The Program Committee has worked diligently with the authors of the papers to assist the editors in the production of this volume and we thank them for their efforts.

Ben Kobler
P C Hariharan

Tenth Goddard Conference on Mass Storage Systems and Technologies

in cooperation with the

Nineteenth IEEE Symposium on Mass Storage Systems

Program Committee

Ben Kobler, NASA Goddard Space Flight Center (Program Committee Chair)
Jean-Jacques Bedet, SSAI
John Berbert, NASA Goddard Space Flight Center
Randal Burns, The Johns Hopkins University
Robert Chadduck, National Archives and Records Administration
Jack Cole, Army Research Laboratory
Bob Coyne, IBM
Jim Finlayson, Department of Defense
Gene Harano, National Center for Atmospheric Research
P C Hariharan, Systems Engineering and Security, Inc.
Jim Hughes, Storage Technology Corporation
John Jensen, National Oceanic and Atmospheric Administration
Merritt Jones, MITRE
Ethan Miller, University of California, Santa Cruz
Reagan Moore, San Diego Supercomputer Center
Matthew O'Keefe, Sistina Software
Bruce Rosen, National Institute of Standards and Technology
Tom Ruwart, Ciprico
Don Sawyer, NASA Goddard Space Flight Center
Rodney Van Meter, Nokia
Richard Watson, Lawrence Livermore National Laboratory

Table of Contents

Tutorials

Perpendicular Recording: A Future Technology or a Temporary Solution, Dmitri Litvinov and Sakhrat Khizroev, Seagate Research 1

OSD: A Tutorial on Object Storage Devices, Thomas M Ruwart, Ciprico, Inc 21

Network Storage 1

IP Storage: The Challenge Ahead, Prasenjit Sarkar andKaladhar Voruganti, IBM Almaden Research Center 35

File Virtualization with DirectNFS, Anupam Bhide, Anu Engineer, Anshuman Kanetkar, Aditya Kini, Calsoft Private Ltd, and Christos Karamanolis, Dan Muntz, Zheng Zhang, HP Research Labs, and Gary Thunquest, HP Colorado 43

Building a Single Distributed File System from Many NFS Servers -or- The Poor Man's Cluster Server, Dan Muntz, Hewlett Packard Labs 59

HSM 1

High Performance RAIT, James Hughes, Charles Milligan and Jacques Debiez, Storage Technology Corporation 65

Conceptual Study of Intelligent Data Archives of the Future, H K Ramapriyan, Steve Kempler, Chris Lynnes, Gail McConaughy, Ken McDonald, Richard Kiang, NASA Goddard Space Flight Center and Sherri Calvo, Robert Harberts and Larry Roelofs, Global Science and Technology, Inc, and Donglian Sun, George Mason University 75

Storage Issues at NCSA: How to get file systems going wide and fast within and out of large scale Linux cluster systems, Michelle Butler, National Center for Supercomputing Applications (NCSA) 93

Potpourri

The Challenges of Magnetic Recording on Tape for Data Storage (The One Terabyte Cartridge and Beyond), Richard H Dee, Storage Technology Corporation 109

Efficient RAID Disk Scheduling on Smart Disks, Tai-Sheng Chang and David H C Du, University of Minnesota 121

Experimentally Evaluating In-Place Delta Reconstruction, Randal Burns, The Johns Hopkins University, Larry Stockmeyer, IBM Almaden Research Center and Darrell D E Long, University of California, Santa Cruz 137

Storage Indexing

Intra-File Security for a Distributed File System, Scott A Banachowski, Zachary N J Peterson, Ethan L Miller and Scott A Brandt, University of California, Santa Cruz 153

Efficient Storage and Management of Environmental Information, Nabil R Adam, Vijayalakshmi Atluri and Songmei Yu, Rutgers University and Yelena Yesha, University of Maryland Baltimore County 165

Indexing and selection of data items in huge data sets by constructing and accessing tag collections, Sebastien Ponce and Pere Mato Vila, CERN and Roger D Hersch, Ecole Polytechnique Lausanne 181

HSM 2

Data Placement for Tertiary Storage, Jiangtao Li and Sunil Prabhakar, Purdue University 193

Storage Resource Managers: Middleware Components for Grid Storage, Arie Shoshani, Alex Sim, Junmin Gu, Lawrence Berkeley National Laboratory 209

Storage Area Networks and the High Performance Storage System, Harry Hulen and Otis Graf, IBM Global Services, and Keith Fitzgerald and Richard W Watson, Lawrence Livermore National Laboratory 225

Network Storage 2

Introducing a Flexible Data Transport Protocol for Network Storage Applications, Patrick Beng T Khoo and Wilson Yong H Wang, Data Storage Institute, National University of Singapore 241

Point-in-Time Copy: Yesterday, Today and Tomorrow, Alain Azagury, Michael E Factor and Julian Satran, IBM Research Lab in Haifa, and William Micka, IBM Storage Systems Group 259

Locating Logical Volumes in Large-Scale Networks, Mallik Mahalingam, Christos Karamanolis, Magnus Karlsson and Zhichen Xu, Hewlett Packard Labs 271

Short Papers

Building a Massive, Distributed Storage Infrastructure at Indiana University, Anurag Shankar and Gerry Bernbom, Indiana University 285

High-density holographic data storage with random encoded reference beam, Vladimir B Markov, MetroLaser, Inc 291

iSCSI Initiator Design and Implementation Experience, Kalman Z Meth, IBM Haifa Research Lab 297

Efficiently Scheduling Tape-resident Jobs, Jing Shi, Chungxiao Xing and Lizhu Zhou, Tsinghua University 305

The Storage Stability of Metal Particle Media: Chemical Analysis and Kinetics of Lubricant and Binder Hydrolysis, Kazuko Hanai and Yutaka Kakuishi, Fuji Photo Film Co Ltd 311

Java and Real Time Storage Applications, Gary Mueller and Janet Borzuchowski, Storage Technology Corporation 317

Vendor Paper

DIR-2000, 1 Gbit/sec Data Recorder for VERA Project, Tony Sasanuma, Sony Broadband Solutions Network Company 327

Index of Authors 331

Perpendicular Recording: A Future Technology or a Temporary Solution

Dmitri Litvinov and Sakhrat Khizroev
Seagate Research
River Park Commons, Suite 550
2403 Sidney Street
Pittsburgh, PA 15203-2116
Tel: +1-412-918-7028
Fax: +1-412-918-7010

Abstract

During the vitally critical times to the future advances in data storage technologies, perpendicular magnetic recording [1,2,3] has attracted a substantial amount of attention as a prime alternative to the technologies in place today [4,5]. As envisioned by the industry and academia leaders, perpendicular recording is the most likely candidate for the technology implemented in the next generations of hard drives. The most competitive virtue of this technology is the fact that while being technically the closest alternative to conventional longitudinal recording, it is capable of extending the (superparamagnetic) density limit [6] beyond what is achievable with longitudinal recording. It is widely believed that perpendicular magnetic recording paradigm will enable to sustain the current great strides in technological advances for the next several generations of magnetic storage solutions.

This paper will cover the basic principles underlying perpendicular recording as well as the challenges associated with implementing the technology [7,8,9,10].

1 Superparamagnetic limit and the need for a new technology

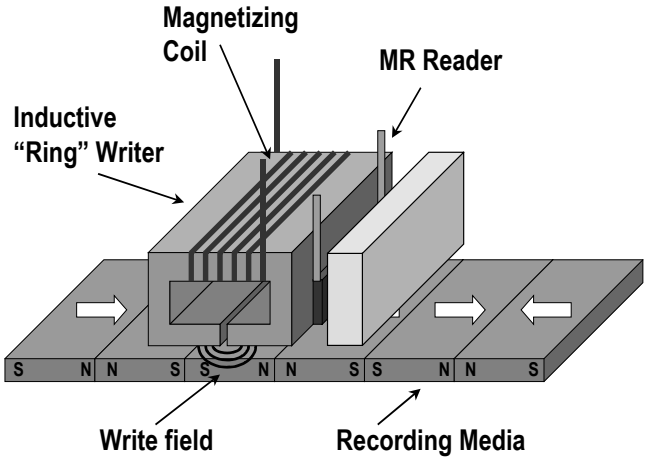


Figure 1. A schematic of a conventional longitudinal recording scheme employed in today's hard drives.

The data on a magnetic recording medium is stored by means of recording a certain spatial variations of the magnetization, where the magnetization variations represent the data. The relation between the data and the magnetization pattern is defined by the encoding scheme used. Figure 1 shows a simplified schematic of a conventional longitudinal recording system. The recording media are engineered such that the

preferred direction of the magnetization, a so-called easy axis, lies in the plane of the recording layer. Using an inductive "ring"-type writer, the magnetization of the grains is aligned along the track in either positive or negative direction. The data is read back using a magnetoresistive element. A change or no change in the magnetization direction at the bit transitions corresponds to a 1 or to a 0, respectively. The lateral dimensions of a bit, i.e. the smallest feature realized in a particular drive design, defines the areal bit density that such a drive supports.

A conventional magnetic medium has granular structure such that each bit consists of several magnetic grains or magnetic clusters. The magnetic clusters/grains are usually shaped irregularly and are randomly packed, as shown in Figure 2a. Consequently, the recording bits and bit transitions are usually not perfect, which is illustrated in Figure 2b. These imperfections lead to noise in the playback signal. The noise is kept below a certain acceptable level by means of including a sufficiently large number of magnetic grains into each bit. The resulting averaging reduces the level of noise. As the areal density increases, the bit size and the size of the grains that constitute the bit, decreases. Typical grains in today's media range from 5 to 15nm.

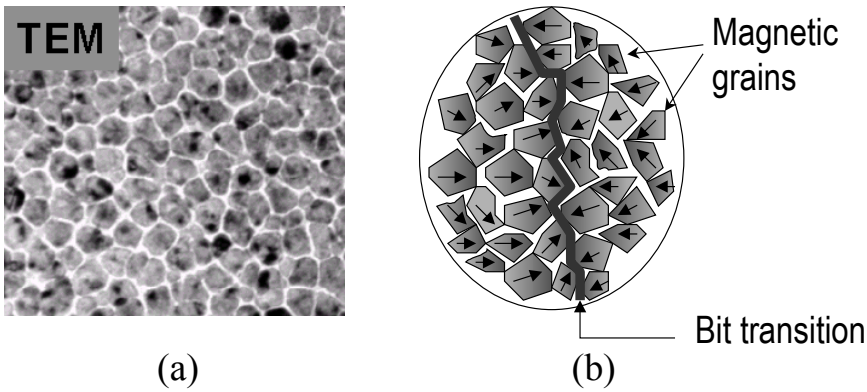


Figure 2. (a) A transmission electron micrograph of a typical granular medium; (b) a schematic of a single bit transition in a granular medium.

One of the critical factors characterizing the reliability of a data storage device is data stability. Various parameters control the stability of the data against the external factors. With respect to the external temperature, which is manifested by thermal fluctuations in the recording media, the magnetic anisotropy energy stored in each magnetic grain is one of the major determinants (assuming that the grains are magnetically independent). The magnetic anisotropy energy approximately defines the amount of energy necessary to reverse the direction of the magnetization of a grain. For a single grain, it is equal to K_UV, where K_U is the magnetic anisotropy energy per unit volume and V is the volume of the grain. For a medium to be thermally stable, the above quantity K_UV should be substantially greater (30-40 times) than the energy of a single quantum of thermal fluctuation, k_BT, where k_B is Boltzman's constant and T is the temperature [6]. As mentioned above, the higher areal densities require smaller grain sizes. It follows that to sustain thermal stability, K_U of a magnetic medium material should increase with the grain size decreases. Unfortunately, as K_U increases, so does the write field necessary to efficiently write onto the medium. In conventional longitudinal recording, the upper limit of the write field that a recording head can generate is equal to $2\pi M_S$ where M_S is the

saturation magnetization moment of the head material. The highest value of $4\pi M_S$ of the materials available today is rapidly approaching what is believed to be a fundamental limit of ~25kGauss. This defines the upper limit of the K_U values that can be employed in a longitudinal medium and, consequently, the maximum areal density achievable with conventional longitudinal recording. It has been predicted that with the materials available today, the highest areal density achievable with conventional longitudinal recording is ~100Gbit/in^2 [5,6].

2 Dodging the superparamagnetic limit … The advantages of perpendicular recording?

Several aspects native to perpendicular recording make it superior to longitudinal recording with respect to the superparamagnetic limit. Among the advantages are higher write-field amplitude and sharper write-field gradients, thicker recording layers, absence of demagnetizing field at bit transitions, higher playback amplitude, etc. The specific nature of these advantages is discussed in detail below.

2.1 Higher write field with sharper side and trailing gradients

Figure 3 shows a comparative schematic of conventional longitudinal and perpendicular recording schemes. While in longitudinal recording, the natural direction of the magnetization, the easy axis, lies in the plane of a recording medium, in perpendicular recording, the easy axis is perpendicular to the plane of a medium. In longitudinal recording, the recording is performed by the fringing fields emanating from the gap region between the write-poles of a conventional "ring"-type recording head. It is the geometry of a longitudinal ring-head that defines the upper limit of the write field of $2\pi M_S$, where M_S is the saturation magnetization of the write-pole material. In perpendicular recording, write field is generated between the trailing pole of a single pole head and a soft underlayer (SUL), a soft magnetic material located below the recording layer. In such geometry, the upper limit of the write field is equal to $4\pi M_S$, which is two times higher than the highest field achievable with a longitudinal ring head.

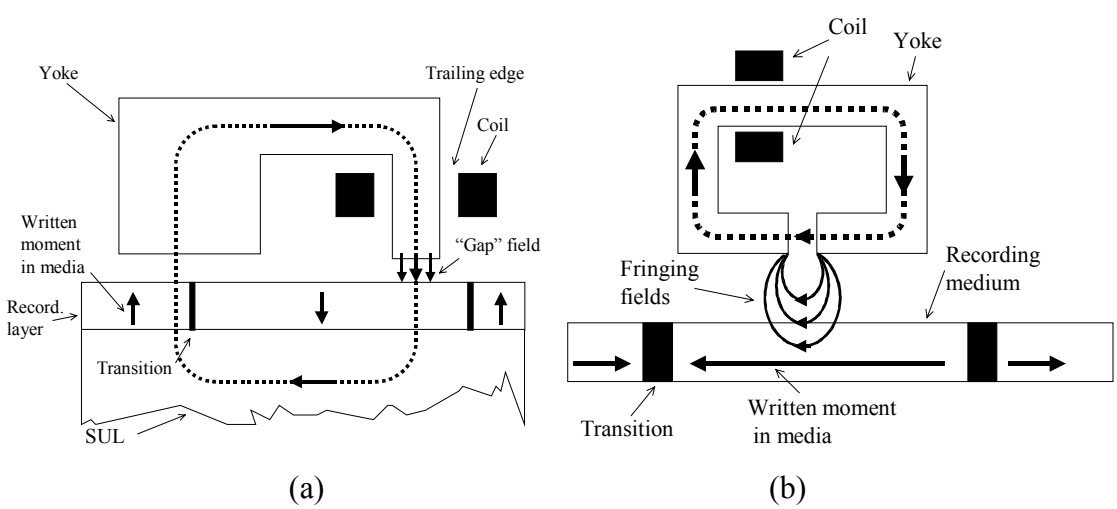


Figure 3. Diagram showing a side cross-section of (a) a typical perpendicular system including a SPH and a double-layer medium with a SUL and (b) a longitudinal system, including a ring-head and a single-layer recording medium.

Higher write efficiency of a perpendicular single-pole recording head in combination with a SUL can ba explained in greater detail as illustrated in Figure 4. It can be shown (the proof of this concept is beyond the scope of this paper [9]) that to evaluate the magnetic fields above the SUL boundary, the SUL can be thought of as a perfect magnetic mirror such that the magnetic field above the SUL boudnary is a superposition of the fields generated by both the magnetic elements above the SUL boundary and by their images located below the SUL boundary. This concept is illustrated in Figure 4, where the SUL is replaced with an image recording head. From this picture it is clear that in perpendicular recording the write process effectively occurs in the gap between the magnetic poles, the real and the image poles, which is in contrast to longitudinal recording where the writing is done by the frinding fields as outlined above. From simple superposition arguments, it is straighforward to show that the in-gap field is equal to $4\pi M_S$ while the highest value of the fringing field is equal to $2\pi M_S$.

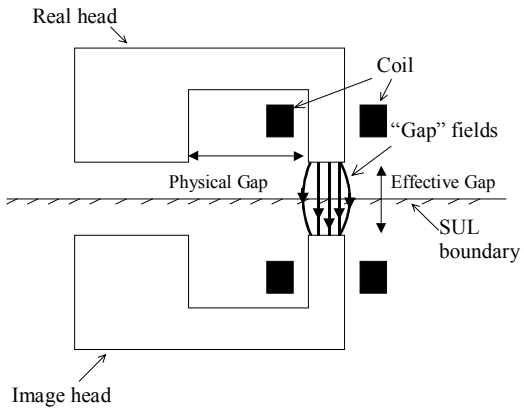
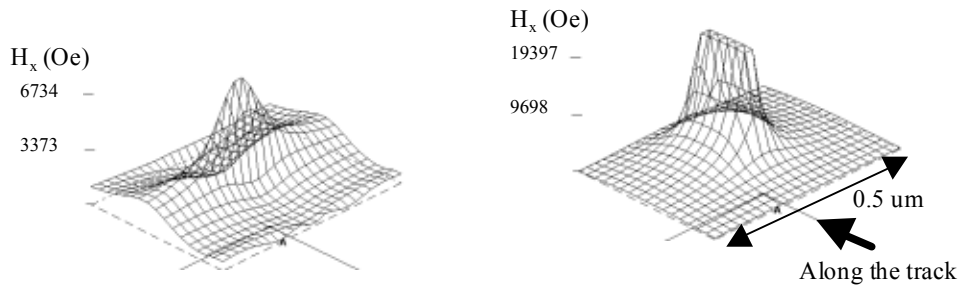


Figure 4 A schematic of the magnetic imaging principle in perpendicular recording using a medium with a soft underlayer.

As shown above, the maximum write field available in perpendicular recording is two times higher than the maximum write available in longitudinal recording. The direct consequence is the ability to write onto a higher anisotropy media (higher K_U). The use of higher anisotropy media materials allows higher areal densities without compromising the thermal stability of the recording data.

The spatial profile of the write field is also more beneficial for achieving higher areal density in perpendicular recording. The side gradients, i.e. the rate at which the field rolls off at the side edges of a recording head, are usually substantially sharper than what one observes in longitudinal recording. This property leads to better-defined tracks with a very narrow erase band. Along with better magnetic alignment of the media (see below), extremely narrow tracks are possible to achieve.



(a) (b)

Figure 5. Longitudinal head field contours and perpendicular head field contours from (a) a longitudinal head with a 150 nm gap and (b) a perpendicular pole head with a pole thickness of 700 nm. The trackwidth is 50 nm in both cases.

The single pole perpendicular write heads used to acquire the experimental data presented in this paper, were made by focused ion-beam (FIB) modification of conventional longitudinal writers [11]. It should be emphasized that the main difference in the design of conventional perpendicular and longitudinal writers is the length of the gap between the magnetic write-poles. In terms of the write process, while in longitudinal recording the writing is done near the gap region, in perpendicular recording, the writing is done by the trailing edge of the trailing pole [12]. Figure 6 shows a state-of-the-art perpendicular recording head manufactured by FIB trimming of a conventional longitudinal write head by increasing the gap length and trimming the trailing pole and the reader to the specified dimensions. Both the trailing pole and the reader are designed for a 60nm track width.

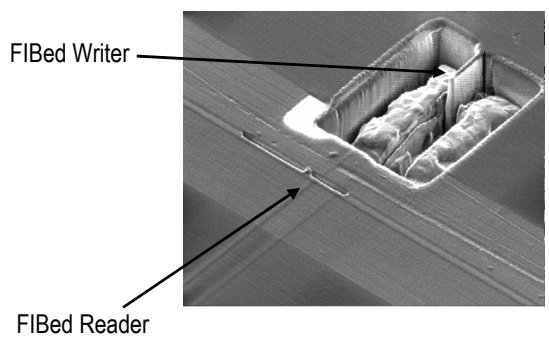


Figure 6. A single pole perpendicular write head made by focused ion-beam etching of a conventional longitudinal ring head. The trailing pole width is 60nm.

2.2 Well aligned media

In conventional longitudinal recording, the easy axes of individual grains are randomly oriented in the plane of a medium. (It should be recalled that the easy axis is the energetically favorable axis/direction along which the magnetization of a grain is aligned in the absence of external magnetic fields.) Thus, in longitudinal recording, a large fraction of the grains forming a bit has their easy axes severely misaligned with the bit magnetization direction. Writing well-defined bit transitions on such randomly oriented media imposes stringent requirements onto the spatial profile of a write-field. If one neglects the imperfections of a bit transition due to the granular nature of a medium, the quality of the bit transition is defined mainly by the write-field profile.

This is drastically different from perpendicular recording, in which the easy axis of each magnetic grain is relatively well aligned in the direction perpendicular to the plain of the medium. Thus, in a perpendicular recording, the magnetization direction of a recorded bit always coincides with the orientation of the easy axes of individual grains that form the bit. Well-defined easy axis orientation relaxes the stringent requirements for the trailing and side write-field gradients necessary to achieve sharp transitions, thus enabling the use of thicker media [10].

The intrinsically better alignment of perpendicular media helps record narrow tracks with well-defined transitions even into a relatively thick recording layer. A MFM image of two adjacent tracks with a 65 nm trackpitch written into a 50 nm thick CoCr recording layer using a 60 nm wide single pole head is shown in Figure 7 [7]. This is equivalent to a track density of ~400ktpi. It should be stressed that the state-of-the-art in longitudinal recording for the track density is ~100ktpi.

The possibility of using thicker recording layers further assists with improving thermal stability.

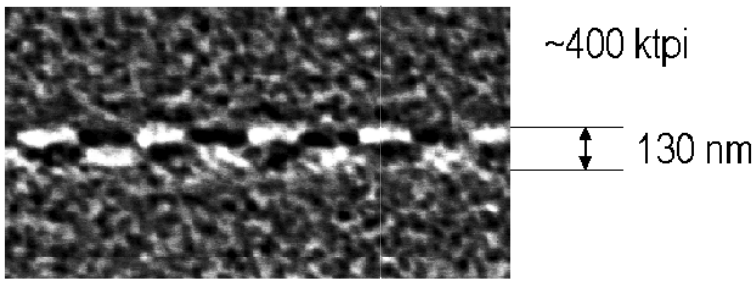


Figure 7. A MFM image of two tracks with a 65 nm trackpitch.

With respect to using well-aligned media, it should be remembered that previously it was shown that, although well-aligned perpendicular media might have a relatively small average angle between the magnetization and the perpendicular recording field, the torque created is still sufficiently large to quickly switch the magnetization [13, 14].

2.3 Absence of demagnetizing fields at bit transitions

One of the major destabilizing factors in longitudinal recording medium is strong demagnetizing field at the bit transition. The destabilizing influence of the demagnetizing field at the bit transitions is easy to see if one notices that the two adjacent bits of opposing magnetization directions repel in a similar way as two bar magnets with the poles of the same polarity, such as north-north or south-south, facing each other. The magnets would try to flip such that the poles of opposite polarities are next to each other. This is illustrated below in Figure 8.

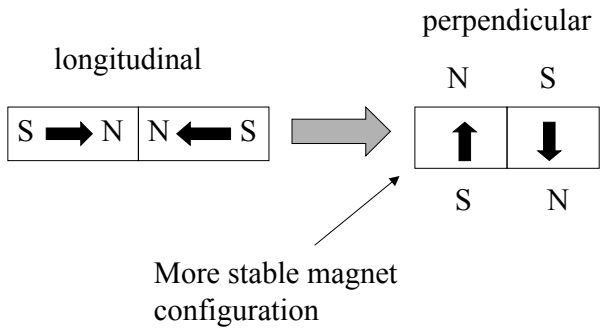
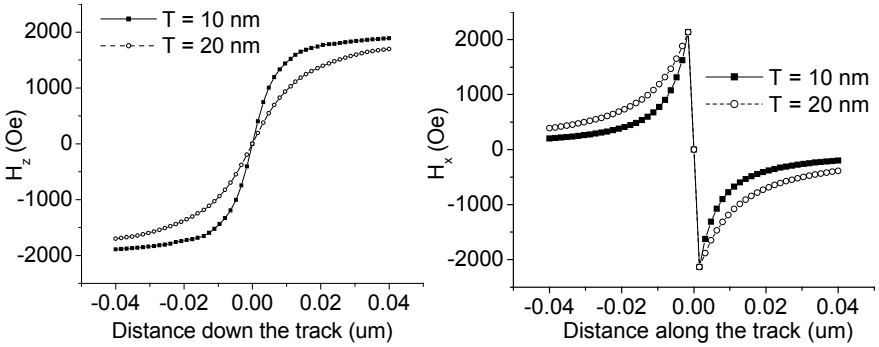


Figure 8. A schematic of the influence of demagnetizing fields in longitudinal and perpendicular media.

The calculated demagnetizing fields for the cases of longitudinal and perpendicular media for a single bit-transition are shown in Figure 9. In the longitudinal recording, high demagnetizing fields at bit-transitions destabilize individual grains leading to a finite transition width. This is opposite to perpendicular recording, in which the demagnetizing fields reach their minima at the bit-transitions, thus promoting ultra-narrow transitions and, consequently, high-density recording.

It can also be noticed that, unlike in longitudinal recording, the demagnetization fields in perpendicular recording decrease as the thickness increases, thus promoting thicker recording layers, which in turn is beneficial for the thermal stability. In this respect, it is common to notice that although perpendicular recording promotes high densities, the stronger influence of the demagnetization fields at lower densities is a disadvantage of perpendicular recording.



(a) (b)

Figure 9. The demagnetization field versus the distance down the track along the central planes of 10 nm and 20 nm thick recording layers for (a) perpendicular and (b) longitudinal recording media.

3 A new system component: soft underlayer challenges and design considerations

One of the key aspects of perpendicular recording that makes it superior to the longitudinal recording with respect to superparamagnetic effects is utilization of media with a SUL. A single-pole head and a medium with a SUL perpendicular recording system enables write fields in excess of 80% of $4\pi M_S$ of the pole head/SUL material. This doubles the fields available in longitudinal recording, thus opening the possibility to

write on substantially higher anisotropy media and leading to better thermal stability. Acting as a magnetic mirror, SUL effectively doubles the recording layer thickness, facilitating substantially stronger readout signals. Also, the effective thickness increase due to the mirroring effects by a SUL leads to the reduction of the demagnetizing fields with a potential to further improve thermal stability.

While the utilization of perpendicular media with a SUL should make it possible to postpone the superparamagnetic limit, the SUL introduces a number of technical challenges. Some of the issues related to the presence of the SUL are discussed below.

3.1 SUL as a major source of noise

Among the technical challenges introduced by the presence of a SUL is the fact that a not properly optimized SUL material can introduce a significant amount of noise into the playback signal. The noise results from the stray field generated by the effective charges resulting from domain walls in the SUL as illustrated in Figure 10.

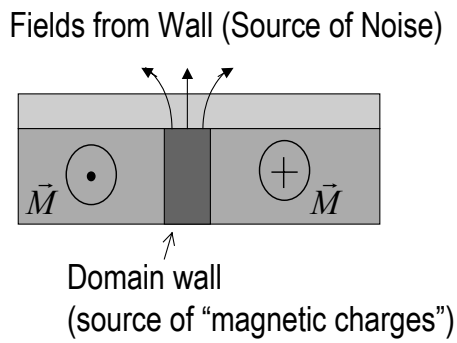


Figure 10. A schematic of the stray fields generated by a SUL

Magnetic biasing of the SUL, i.e. forcing the SUL into a single magnetic domain state, allows to minimize the SUL noise. The biasing can be achieved either by application of an external magnetic field or by engineering a SUL material with a built-in biasing field. Figure 11 shows a schematic of the experimental setup to study the effect of magnetic biasing of the SUL on the noise. The magnetic biasing was achieved using two NdFeB permanent magnets placed in the vicinity of the media. The placement of the magnets was such that it allowed achieving complete saturation of the SUL underneath the reader. Special care was necessary to arrange the magnets sufficiently far from the recording head ~2cm away in order not to affect the properties of the read element.

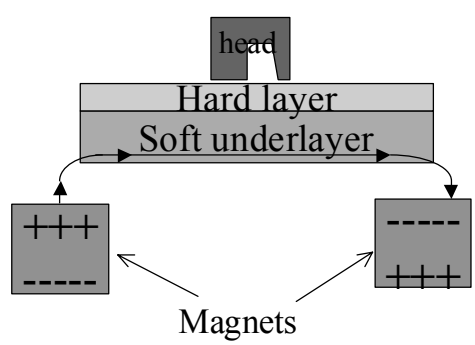
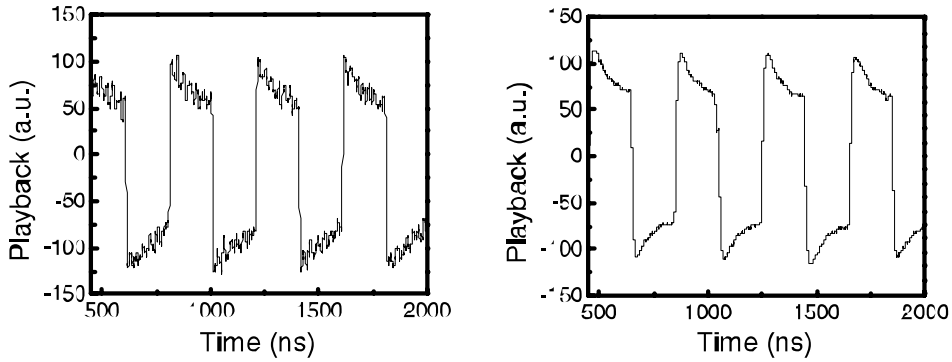


Figure 11. A schematic of experimental setup to magnetically bias SUL film.

Figure 12 shows the playback signals from the two media with as deposited non-biased (a) and magnetically biased (b) SUL's. A substantial level of noise attributed to presence of a large number of domain walls (confirmed by magnetic force microscopy) in the SUL can be seen in Figure 12a. A drastic reduction of the noise (by at least 10dB) is clearly observed in Figure 12b where the SUL is magnetically biased.



(a) (b)

Figure 12. Playback signal from two media with different SUL's. (a) SUL with a large number of stripe domains. The presence of stripe domains was confirmed using magnetic force microscopy. (b) Biased SUL with domain walls swept out from the SUL material.

The magnetic biasing saturates SUL film forcing it into a pseudo-single domain state effectively sweeping the domain walls out of the SUL material. This results in elimination of the SUL noise.

3.2 SUL magnetic moment

To properly design a perpendicular recording system that utilizes a medium with a SUL, it is critical to choose an appropriate SUL material. As illustrated in Figure 13, if the magnetic moment of a SUL material is lower than the magnetic moment of the recording pole tip, saturation of the SUL underneath the pole tip can occur.

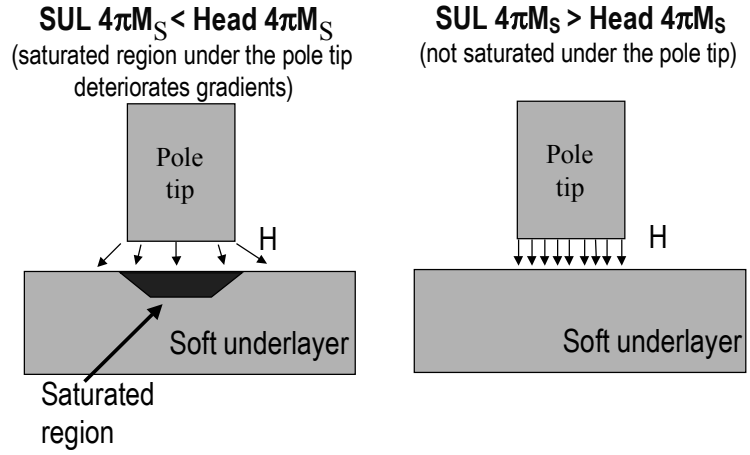


Figure 13. A schematic illustrating the saturation effect in the SUL is the magnetic moment of a SUL is lower than the magnetic moment of the write pole tip.

The results of boundary element modeling for two different head/SUL combinations are presented in Figure 14. It can be noticed that it is possible to generate strong recording fields with the magnitude approaching $4\pi M_S$ of the pole tip even if the SUL has a lower magnetic moment than the pole tip. However, saturation of the SUL will lead to a substantial deterioration of the trailing field gradients. The trailing gradients in the case of the Permalloy based SUL are substantially worse than the trailing gradients in the case when a FeAlN based SUL is used.

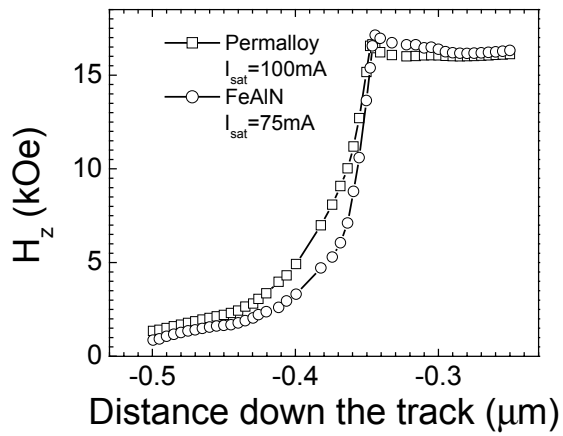


Figure 14. Trailing fields from a single pole perpendicular write head made out of FeAlN ($4\pi M_S$ =20kG) for FeAlN and Permalloy ($4\pi M_S$ =10kG) SUL's.

It follows that if high moment materials are used for write heads, e.g. CoFeB, FeAlN, etc., the moment of the SUL material should match or exceed the moment of the pole tip material.

3.3 SUL thickness

Another important issue related to the optimized design of a SUL is the SUL thickness. Using simple considerations of magnetic flux conservation, the minimum thickness required for the SUL to function properly is given by

$$t_{\text{soft underlayer}} \geq \frac{1}{2} \frac{M_{\text{S pole tip}}}{M_{\text{S soft underlayer}}} w_{\text{pole tip}},$$

where the $w_{\text{pole tip}}$ is the width of the write pole tip, i.e. the dimension of the write pole tip defining the track width. The evaluation of the above equation for the case of 100Gbit/in^2 areal density and 4:1 bit aspect ratio, i.e. a 160nm wide pole tip, and the same pole tip and SUL materials, gives the lower boundary on the SUL thickness of 80nm. It should be stressed that this thickness is substantially smaller than the minimum required thickness often quoted in the literature of hundreds of nanometers to several microns.

This important observation needs to be strongly emphasized. Due to materials properties, the mentioned above problem of SUL noise becomes increasingly aggravated with the increasing thickness of the SUL.

3.4 SUL influence on the resolution of a perpendicular recording system

An additional challenge that the presence of a SUL imposes is potential deterioration of the system resolution. During reading from a medium with a SUL, due to the magnetic imaging properties of the SUL, the resolution can get distorted if the separation between the ABS and the SUL (sum of the recording layer thickness and the flying height) is comparable to the reader thickness.

This phenomenon is clearly illustrated in the calculated [15] PW50 and the playback signal versus the underlayer to the ABS distance, shown in Figure 15. PW50 is the physical width of a single transition, the measure of the spatial resolution of a recording system. In these calculations, a fixed recording layer thickness of 10 nm was assumed, and spacing between the bottom side of the recording layer and the underlayer was varied from zero to some finite values. For comparison, the dotted straight lines indicate the values for the case when there is no underlayer. It can be clearly seen that the resolution of the modeled recording system substantially deteriorates at certain values of the ABS-to-SUL spacing. This suggests that a special care has to be taken to properly optimize the system's resolution.

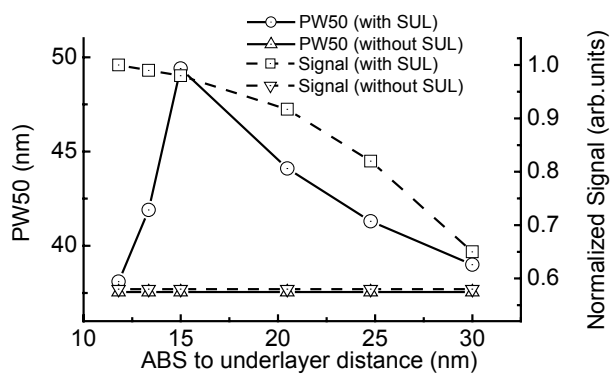


Figure 15. PW50 and the normalized playback vs. the ABS to underlayer spacing. 30 nm GMR element and a 70 nm shield-to-shield spacing are assumed.

Although, in a properly designed system this resolution distortion can be almost completely eliminated, it causes the resolution of a typical read head in a system with an underlayer to be at most as good as the resolution of an equivalent head in a system without an underlayer. It should be noted, however, the underlayer definitely increases the playback signal, which is desirable at high areal densities.

4 Playback: new signal processing schemes

One of the drastic differences between perpendicular and longitudinal recording is the difference in playback signals. To help understand the basic difference in the playback process between longitudinal and perpendicular recording, schematic diagrams of the stray fields emanating from a longitudinal medium and perpendicular media without and with a SUL are shown in Figure 16, respectively. As can be noticed, in the longitudinal case, the stray fields emanate only from the transitions, with the fields near the transitions oriented perpendicular to the disk plane. On the contrary, in the perpendicular cases, the stray field emanates from the effective magnetic "charges" at the top and effective (due to

a SUL) bottom surfaces of the recording layer, with the field near the transitions oriented parallel to the disk plane.

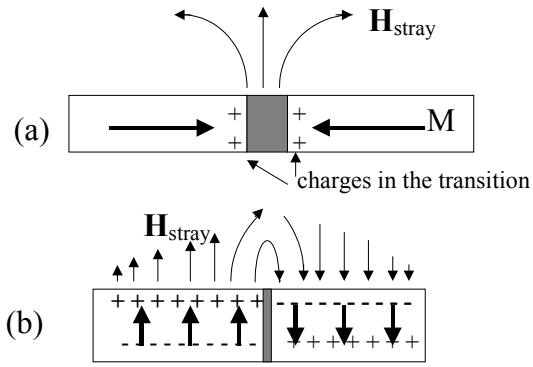


Figure 16. Diagrams showing the sources of stray fields in the case of (a) longitudinal recording, and (b) perpendicular recording.

As a result of the different magnetic "charge" distributions, the playback waveform differ drastically between longitudinal and perpendicular recording schemes. It is illustrated in Figure 17 where typical low-density playback waveforms are shown for both perpendicular and longitudinal recording.

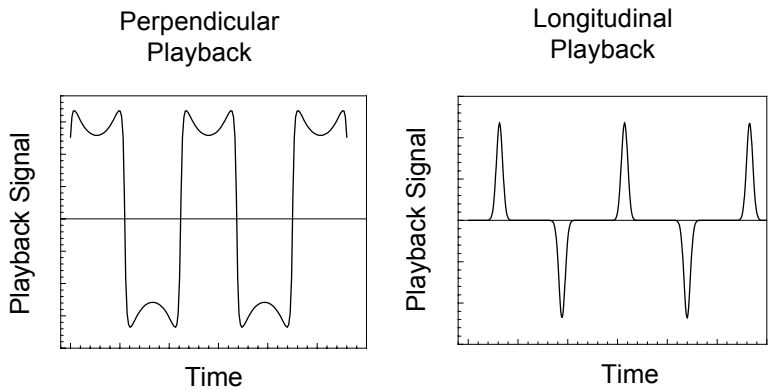


Figure 17. Typical playback waveforms for perpendicular and longitudinal recording schemes.

The shown above waveforms for perpendicular and longitudinal recording schemes outline major difference between perpendicular and longitudinal recording. While in longitudinal recording the signal is present only at bit transitions, in perpendicular recording the signal is read not only from a bit transition but also from across the whole bit area. It is possible to differentiate the perpendicular playback signal to make it similar to the playback signal in longitudinal recording. However, it should be remembered that differentiate perpendicular playback is only similar but not identical to longitudinal playback. The difference arises in the absence of a transition when a longitudinal playback signal is equal to zero while a differentiated perpendicular playback is, although relatively small in amplitude, but is still finite.

It should be stressed that while not entirely suited to be processed by conventional longitudinal channels, perpendicular playback clearly contain more information than typical longitudinal waveforms, in which the signal arrives only from transitions. This property could potentially be used to advantage in future channel designs.

5 New materials challenges

While the requirements for the head materials used in perpendicular recording are similar to the head materials used in longitudinal recording, the major differences exist with respect to media materials. A typical perpendicular medium consists of two magnetically active layers: a hard layer and a SUL (See Figure 18). A hard layer in a perpendicular medium has rather different magnetic properties from a hard layer utilized in conventional longitudinal recording. It should also be noted that there is no analog to a SUL in longitudinal recording. The requirements for these two layers are outlined below.

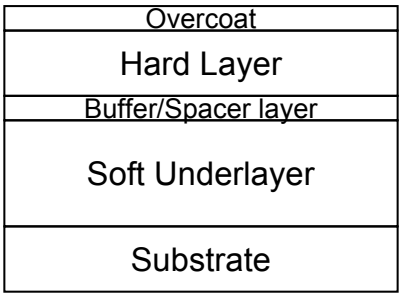


Figure 18. A schematics of a typical perpendicular medium.

5.1 Hard layer materials

The primary approach to the design of a perpendicular recording layer is in many ways similar to the design of a conventional longitudinal recording layer. All the media in use today has granular structure, i.e. made of polycrystalline materials. Major goals inherent to both longitudinal and perpendicular recording layer development are small grain size, small grain size distribution, texture control, optimization of the inter-granular exchange de-coupling, etc.

A large variety of today's perpendicular magnetic recording layer types can be clearly divided into the two major categories: 1) Alloy based media, such as CoCr-alloys[16, 17], and 2) media based on magnetic multilayers, such as Co/Pt, Co/Pd or others[18, 19]. Figure 19 contrasts the major difference between alloy and multilayer media. In alloy media, the magnetic anisotropy is controlled by magnetic crystalline anisotropy. The alloy media are usually highly textured to insure well-defined magnetic easy axis [20]. In magnetic multilayers, the magnetic anisotropy is controlled by interfacial effects between a magnetic layer, such as Co, and a highly polarizable spacer layer, such as Palladium or Platinum. In contrast to alloy media, this set of materials as used in perpendicular media usually possesses a very weak texture.

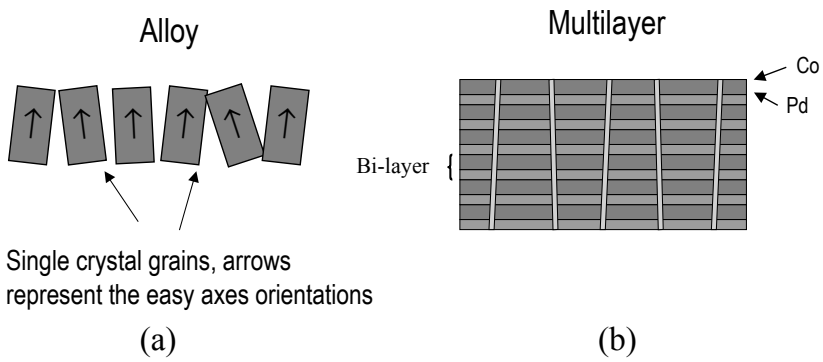
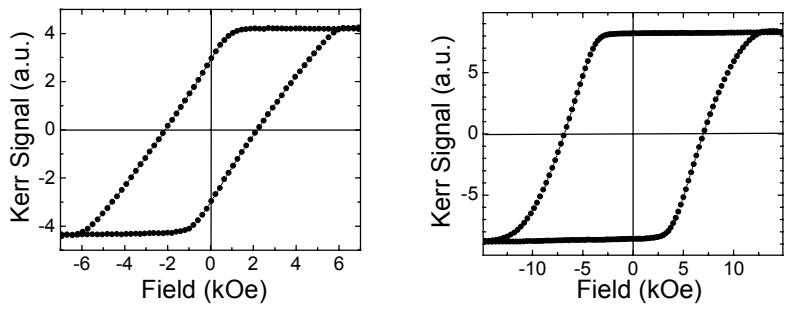


Figure 19. A schematic representation of major microstructural differences

Material-wise, perpendicular CoCr-based alloy recording layers are similar to conventional longitudinal CoCr-based media, with the major difference being the orientation of the magnetic easy axis. Therefore, a significant amount of information accumulated in the course of the longitudinal media development can be used to control the critical parameters such as the grain size and the inter-granular exchange coupling. At the same time, CoCr-based perpendicular media have some open issues. For example, it is not clear yet if it is possible to make a CoCr-based medium with sufficiently high anisotropy to avoid superparamagnetic instabilities at ultra-high areal densities. It also has proven to be difficult to make CoCr-alloy based perpendicular recording layers with a remanent squareness of 1. The remanent squareness is defined as a ratio between the remanent magnetization, the value of magnetization on a M-H loop at H=0, and the saturation magnetization, the maximum value of magnetization. It is believed that a remanent squareness of 1 is necessary for low-density bit pattern stability. Also, a remanent squareness of less than 1 can lead to substantial amounts of DC noise. Various magnetic alloys such as L10 phases of FePt, CoPt, etc. are being studied as higher anisotropy alternatives for the recording layer.

The magnetic multilayer based recording layers typically have significantly larger anisotropy energies (Coercive fields of above 15 kOe have been reported.) and are thus promising to be extendable to significantly higher recording densities. Another advantage of the magnetic multilayers is the fact that typically these materials have a remanent squareness of 1.

To compare basic magnetic properties of CoCr-alloy and mutlilayer based recording layers, typical M-H loops by a Kerr magnetometer for a 50 nm thick perpendicular CoCr thin-film and a 52 nm thick Co/Pd structure (a stack of 40 sets of adjacent 3 and 10 Angstrom thick layers of Co and Pd, respectively) are shown in Figure 20a and b, respectively. It can be noticed that in addition to the remanent squareness of 1, the Co/Pd structure exhibits nucleation fields in excess of 3kOe, a useful characteristic to avoid data self-erasure due to stray fields. Meanwhile, the CoCr material shown in Figure 20a has a squareness of 0.75. The CoCr and Co/Pd recording layers have coercive fields and magnetizations of approximately 3 kOe and 9 kOe and 300 emu/cc and 200 emu/cc, respectively.



(a) (b)

Figure 20. An M-H loop of a 50nm thick (a) CoCr-alloy layer and (b) Co/Pd multilayer.

The direct consequence of remanent squareness less than 1 is shown in Figure 21, which compares the spectral SNR distributions for the two media types. The CoCr medium exhibits a significant amount of noise at lower linear densities. This is mainly due to the fact that the dominant contribution to the noise at low linear density in the CoCr-based medium comes from the DC noise which results from the relatively low value of remanent squareness, as described below in more detail.

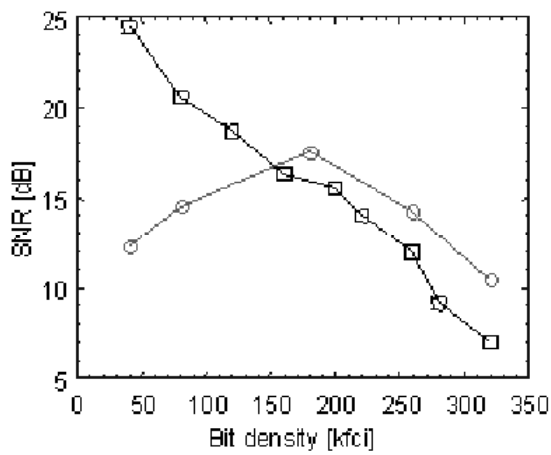


Figure 21. SNR versus the linear density for a CoCr-alloy (hollow circles) and a Co/Pd multilayer (hollow squares).

5.2 High anisotropy SUL materials

Several design guidelines for SUL's were discussed above including thickness requirement and magnetic moment requirement. An additional parameter, which is critical to achieve optimized performance of a SUL in a perpendicular recording system, is magnetic anisotropy of the SUL material. The dynamic properties [21, 22] and influence of a SUL on system's resolution [23] are affected by the value of the anisotropy field. The latter is illustrated in Figure 22, where the playback versus the linear density (roll-off) curves are shown for identical perpendicular recording systems with different SUL materials. The explanation of the quantum-mechanical nature of this effect is beyond the scope of this paper. However, it should be mentioned that the deterioration of the system's resolution arises from inability of lower anisotropy SUL materials to perfectly respond to spatially-fast varying magnetization patterns in the recording layer.

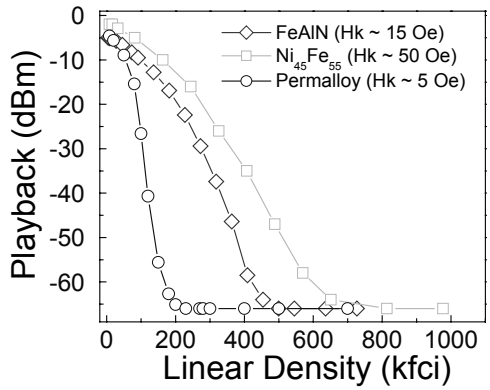


Figure 22. Playback roll-off curves for perpendicular recording media with identical recording layer but different SUL's. The extent of the roll-off curves to higher linear densities for higher anisotropy SUL indicates the advantage of using high anisotropy SUL materials.

6 How far perpendicular recording will take us and what will come next?

It should be emphasized that although perpendicular recording allows to surpass the superparamagnetic limit of longitudinal recording, there exists a superparamagnetic limit native to perpendicular recording as well. A number of factors such as the availability of higher write fields, possibility of using thicker well-aligned media, and the absence of demagnetizing fields at bit transitions aid in promoting thermally stable media to substantially higher areal densities. However, it has been shown that with all factors taken into account, the maximum areal density achievable with perpendicular recording scheme in development today is 500-1000 Gbit/in^2 [5,24,25]. Once the perpendicular magnetic recording reaches its superparamagnetic limit, a new wave of technological innovations will have to take place.

As mentioned in the beginning of this text, the foremost fundamental reason for the existence of the superparamagnetic limit is the head materials constraint imposing the limitation on the available head field that limits the utilization of higher anisotropy media. Among the potential successors of perpendicular recording is heat-assisted magnetic recording (HAMR) [26], in which the anisotropy of a recording medium is temporarily reduced during the write process. In HAMR schemes, an additional element to be incorporated in the design of a recording system is a source of heat (envisioned as an ultra-small light source) to locally increase the temperature of the recording medium. The increase of the medium temperature leads to the decrease of the medium coercivity enabling the writing with relatively small magnetic fields.

Additionally, patterned media can be utilized to further extend the limits of magnetic recording [26]. In a patterned medium, the location and the size of the magnetic features are pre-determined by the medium manufacturing process. Elimination of the element of randomness characteristic to today's polycrystalline recording media is a clear advantage of the patterned medium approach. However, for such a medium to become a serious contender to replace conventional alloy or multilayer media, an economically viable manufacturing process will have to be developed [27,28].

It should be emphasized that due to the advantageous nature of perpendicular recording in promoting extremely high areal bit densities (high write field amplitude, well aligned medium, sharp field gradients, absence of demagnetizing field at transitions, etc.), the future technologies such as mentioned above HAMR and recording on a patterned medium, are likely to be developed as extensions of perpendicular magnetic recording schemes [26] rather than to be based on conventional longitudinal recording.

References

[1] S. Iwasaki and Y. Nakamura, ”An analysis for the magnetization mode for high density magnetic recording,“ IEEE Trans. Magn., vol. 13, p.1272, 1977.

[2] George J. Y. Fan, “Analysis of a practical perpendicular head for digital purposes,” JAP, Vol. 31 (5), p. 402S, 1960.

[3] W. Cain, A. Payne, M. Baldwinson, R. Hempstead, “Challenges in the practical implementation of perpendicular magnetic recording,” IEEE Trans. Magn., Vol. 32 (1), p. 97, 1996.

[4] D.A. Thompson, “The role of perpendicular recording in the future of hard disk storage,” J. Magn. Soc. Of Japan 21, Supplement No. S2, p. 9, 1997.

[5] N. H. Bertram and M. Williams, “SNR and density limit estimates: a comparison of longitudinal and perpendicular recording,” IEEE Trans. Magn., vol 36(1), p. 4, 1999.

[6] S.H. Charap, “Thermal Stability of Recorded Information at High Densities,” IEEE Trans. Magn., Vol. 33(1), p. 978, 1997.

[7] S. Khizroev, M.H. Kryder, and D. Litvinov, “Next generation perpendicular system,” Vol. 37(4), p. 1922, 2001.

[8] D. Litvinov, M.H. Kryder, and S. Khizroev, “Recording physics of perpendicular media: soft underlayers,” J. Magn. Magn. Mater., **232** (1-2), 84-90, 2001.

[9] S. Khizroev and Y.-K. Liu and K. Mountfield and M. H. Kryder and D. Litvinov, “Physics of Perpendicular Recording: Write Process,” J. Magn. Magn. Mater., *in press,* 2002.

[10] D. Litvinov, M.H. Kryder, and S. Khizroev, “Recording physics of perpendicular media: hard layers,” J. Magn. Magn. Mater., *in press*, 2002.

[11] S.K. Khizroev and M.H. Kryder and Y. Ikeda and K. Rubin and P. Arnett and M. Best and D.A. Thompson, “Recording heads with trackwidths suitable for 100Gbit/in^2,” IEEE Trans. Magn., Vol. 35, p. 2544, 1999.

[12] D. Litvinov, J. Wolfson, J. Bain, R. Gustafson, M.H. Kryder, and S. Khizroev, “The role of the gap in perpendicular single pole heads,” to be presented at the 1st North American Perpendicular Magnetic Recording Conference in Coral Gables, Florida, January 2002.

[13] A. Lyberatos, S. Khizroev, and D. Litvinov, "High speed coherent switching of fine grains," IEEE Trans. Magn., Vol. 37(4), p. 1369, 2001.

[14] A. Lyberatos, S. Khizroev, and D. Litvinov, "Thermal effects in high-speed switching in perpendicular media," to be presented at the 1st North American Perpendicular Magnetic Recording Conference in Coral Gables, Florida, January 2002.

[15] S. Khizroev, J. Bain, and M.H. Kryder, "Considerations in the design of probe heads for 100 Gbit/in^2 recording density," IEEE Trans. Magn., Vol. 33(5), p. 2893, 1997.

[16] J.K. Howard, "Effect of nucleation layers on the growth and magnetic properties of CoCr and CoCr-X films," J. Vac. Sci. Techn., Vol 4(6), p. 2975, 1986.

[17] B. Lu, T. Klemmer, S. Khizroev, J.K. Howard, D. Litvinov, A.G. Roy, and D. Laughlin, "CoCrPtTa/Ti perpendicular media deposited at high sputtering rate," IEEE Trans. Magn., Vol. 37(4), p. 1319, 2001.

[18] T.K. Hatwar and C.F. Brucker, "Coercivity enhancement of Co/Pt superlattices through underlayer microstructure modification," IEEE Trans. Magn., Vol 31(6), p. 3256, 1995.

[19] D. Litvinov, T. Roscamp, T. Klemmer, M. Wu, J.K. Howard, and S. Khizroev, "Co/Pd Multilayer Based Recording Layers for Perpendicular Media," MRS Proceedings, T3.9, Vol. 674, 2001.

[20] D. Litvinov, H. Gong, D. Lambeth, J.K. Howard, and S. Khizroev, "Reflection high-energy electron diffraction based texture determination: magnetic thin films for perpendicular media," J. Appl. Phys., Vol. 87 (9), p. 5693, 2000.

[21] D. Litvinov, R. Chomko, J. Wolfson, E. Svedberg, J. Bain, R. White, R. Chantrell, S. Khizroev, "Dynamics of Perpendicular Recording Heads," IEEE Trans. Magn., Vol. 37(4), p. 1376, 2001.

[22] J. Wolfson, J. Bain, S. Khizroev, and D. Litvinov, "Dynamic Kerr imaging of soft underlayers in perpendicular recording," presented at MMM, Seattle, Washington, November 2001.

[23] D. Litvinov, R.M. Chomko, L. Abelmann, K. Ramstock, G. Chen, S. Khizroev, "Micromagnetics of a soft underlayer," IEEE Trans. Magn., Vol. 36(5), p. 2483, 2000.

[24] R. Wood, "Recording Technologies for Terabit per square inch Systems," presented at the 1st North American Perpendicular Magnetic Recording Conference, Coral Gables, Florida, January 2002, to be published in IEEE Transactions on Magnetics, July 2002.

[25] M. Mallary, A. Torabi, and M. Benakli, "1Tb/in2 Perpendicular Recording Conceptual Design," presented at the 1st North American Perpendicular Magnetic Recording Conference, Coral Gables, Florida, January 2002, to be published in IEEE Transactions on Magnetics, July 2002.

[26] M.H. Kryder, "Perpendicular Recording - Its Window of Opportunity and What will Replace It," presented at the 1st North American Perpendicular Magnetic Recording Conference, Coral Gables, Florida, January 2002.

[27] M. Albrecht, C.T. Rettner, S. anders, T. Thompson, M.E. Best, A. Moser, and B.D. Terris, "Recording Properties of Patterned Co70Cr18Pt12 Perpendicular Media," presented at the 1st North American Perpendicular Magnetic Recording Conference, Coral Gables, Florida, January 2002, to be published in IEEE Transactions on Magnetics, July 2002.

[28] J. Moritz, S. Landis, B. Dieny, A. Lebib, Y. Chen, B. Rodmacq, M. Belin, J. Fontaine, C. Donnet, and J.P. Nozieres, "Patterned Media Using Pre-Etched Si Wafers Fabricated by Nano-Imprint and e-beam Lithography," presented at the 1st North American Perpendicular Magnetic Recording Conference, Coral Gables, Florida, January 2002.

OSD: A Tutorial on Object Storage Devices

Thomas M. Ruwart
Advanced Concepts
Ciprico, Inc.
Plymouth, MN 55441
tmruwart@ciprico.com
Tel: +1-612-850-2918
Fax: +1-763-551-4002

Abstract

Ever since online digital storage devices were first introduced in the late 1950's and early 1960's, the various functions key to storing data on these devices have been slowly migrating into the devices themselves. Early disk drives would send analog signals from the read/write head to a physically separate box that would deserialize and frame data into bytes. This data would then be sent to other processors to perform redundancy checks and data transmission to the requesting computer system. As engineers were able to fit more functionality into smaller spaces at reasonable costs, these key functions were migrated into the disk drive itself to the point where we now have an entirely self-contained unit complete with all the electronics that used to fill a small room.

However, even with the integrated advanced electronics, processors, and buffer caches, these disk drives are still relatively "dumb" devices. They essentially perform only two functions: read data and write data. Furthermore, the disk drives do not know anything about the data that they are storing. Things such as content, structure, relationships, quality of service, …etc. are all pieces of information that are external to the disk drive itself. The basic premise of Object Storage Devices is that the disk drive or, more generically, the storage device, can be a far more useful device if it had more information about the data it manages and was able to act on it.

This paper is intended to provide the reader with an overview of OSD, its history, its current state, and possible futures. It begins by presenting a brief history of Object Storage Devices and then discusses why OSD is an important step in the evolution of storage technologies in general. The basic OSD architecture is compared with current Direct Attached Storage (DAS), Storage Area Network (SAN), and Network Attached Storage (NAS) architectures in terms of management, device and data sharing, performance, scalability, and device functionality. Finally, the current status of OSD and related roadmaps are presented as a frame of reference.

Brief History of OSD

The most active work on OSD has been done at the Parallel Data Lab at Carnegie Mellon University (www.pdl.cmu.edu) originally under the direction of Garth Gibson [1,4,5,6,8]. This work focused on developing the underlying concepts of OSD and two closely related areas called Network Attached Secure Disks (NASD) and Active Disks. Other work has been done at the University of California at Berkeley [Keeton], the Universities of California Santa Barbara and Maryland [3], as well as Hewlett Packard Labs [7,9],

Seagate Technology, and Intel Labs. Topics covered by these early pioneers can be broken down into two main categories: OSD architecture and applied OSD concepts. The basic OSD architecture defined to date specifies a set of object functions that can be implemented over any transport (TCP/IP, SCSI, VI, …etc.) but the initial transport will be SCSI for the sake of ubiquity.

Motivation behind OSD

As disk drives and other types of storage devices become denser and more numerous the block-level methods used to access and manage them are reaching the limits of their scalability. OSD is a protocol that defines higher-level methods of communicating the creation, writing, reading, and deleting of data objects as well as other related functions for getting and setting object attributes. OSD is a level higher than a block-level access method but one level below a file-level access method. OSD is not intended to replace either block-level or file-level access methods but rather to add a needed layer of abstraction that sits between them. It is a technology intended to help make existing and future data storage protocols more effective in several areas that include:

- Storage Management
- Security
- Device and Data Sharing
- Storage Performance
- Scalability
- Device Functionality

These areas are becoming more critical to the success of storage *users* as well as the storage *vendors* who are increasingly concerned over ways to differentiate their products. It is quite possible that the OSD architecture will provide both the users and vendors with a highly flexible base on which to build new storage systems that can accommodate each of these areas more effectively than trying to extend the current block-based or file-based protocols.

DAS/SAN/NAS Basic Architectures

There are three basic storage architectures commonly in use today. These are Direct Attach Storage (DAS), Storage Area Networks (SAN), and Network Attached Storage (NAS). Each of these is used to solve problems specific to a particular application or installation. Each has its strengths and weaknesses.

	DAS	**SAN**	**NAS**
Storage Management	High/low	High	Medium
Security	High	Medium	Low
Device and Data Sharing	Low	Medium	High
Storage Performance	High	High	Low
Scalability	Low	Medium	Medium
Device Functionality	Low	Low	Medium

Table 1. Capability assessment based on Technology

The DAS/SAN/NAS architectures and how they scale from a single subsystem to multiple systems are described in diagrams 1-3. Diagrams 4 and 5 show the basic architecture for OSD and the scaling thereof.

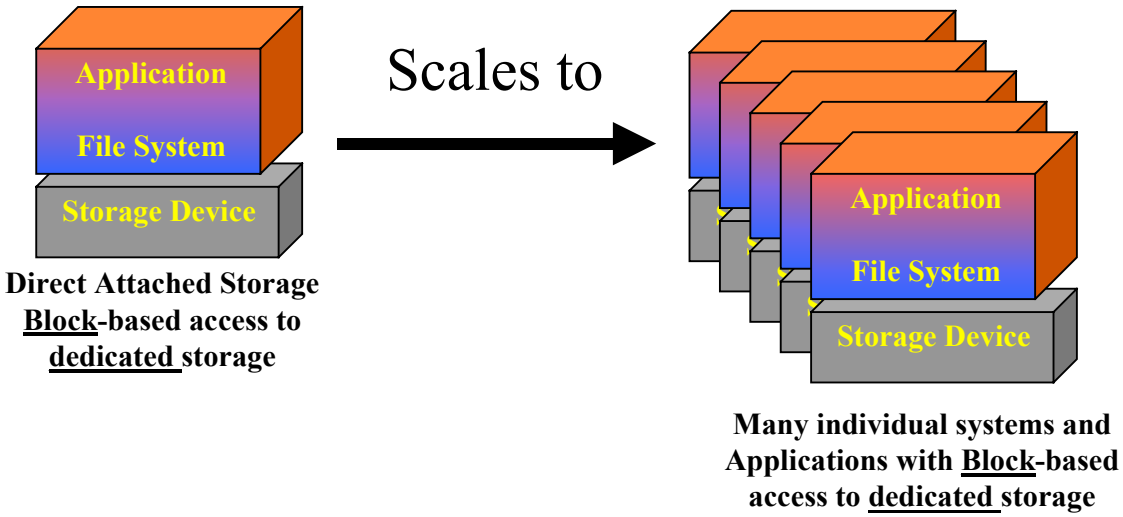


Diagram 1. A single DAS scaling to multiple DAS systems. Each DAS system could conceivably add more storage devices but this is intended to show that when the limit of storage device connectivity is reached on a DAS system, the DAS system must be replicated.

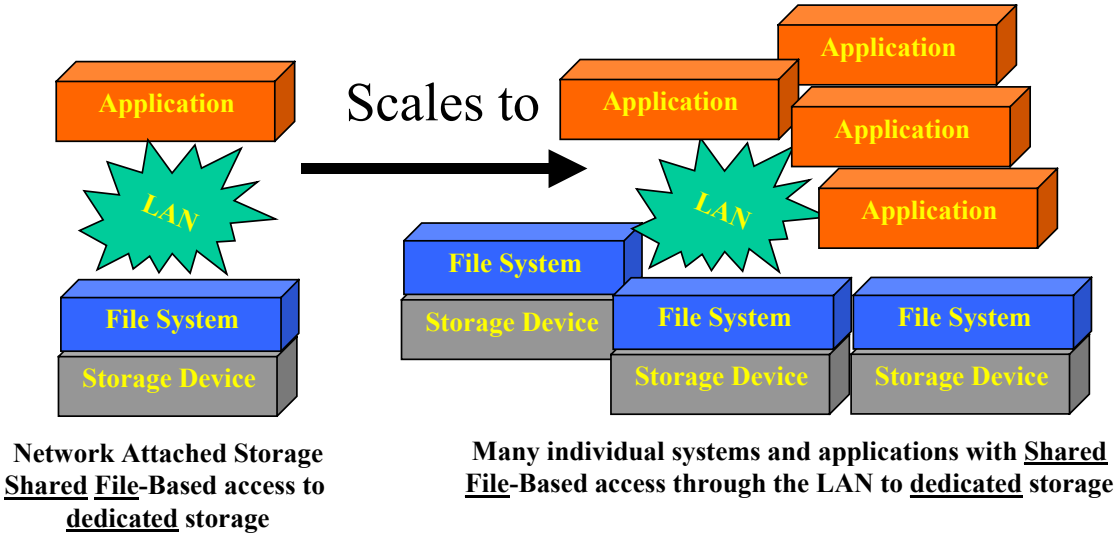


Diagram 2. A single NAS scaling to multiple NAS and multiple application (clients). Note that the NAS boxes themselves can increase in capacity and that they scale in number independently from the application systems (clients).

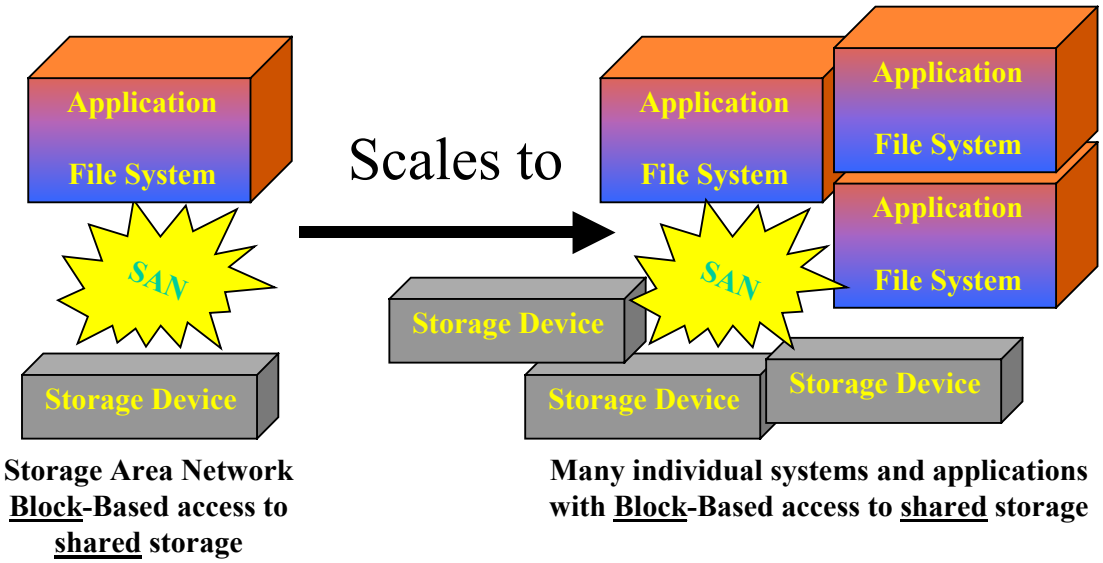


Diagram 3. A single SAN scaling to a larger SAN. Note that the storage devices and application (client) systems scale independently. There is implied device sharing and data sharing in this diagram.

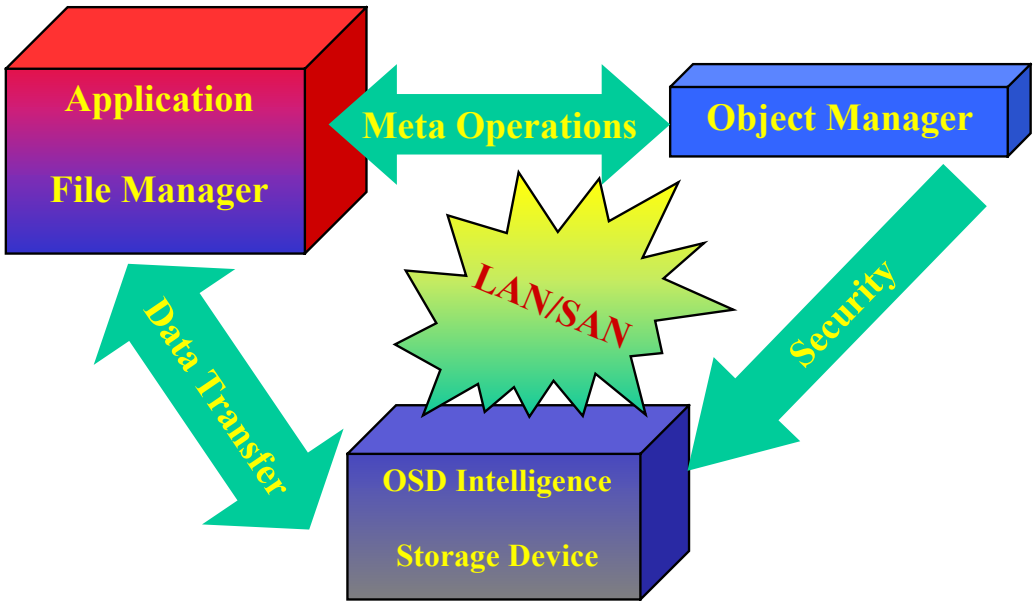


Diagram 4. A basic OSD architecture. Unlike DAS/SAN/NAS the Object Manager is a separate entity from the OSD and the application system (client). The transport for OSD can be either a LAN or a SAN.

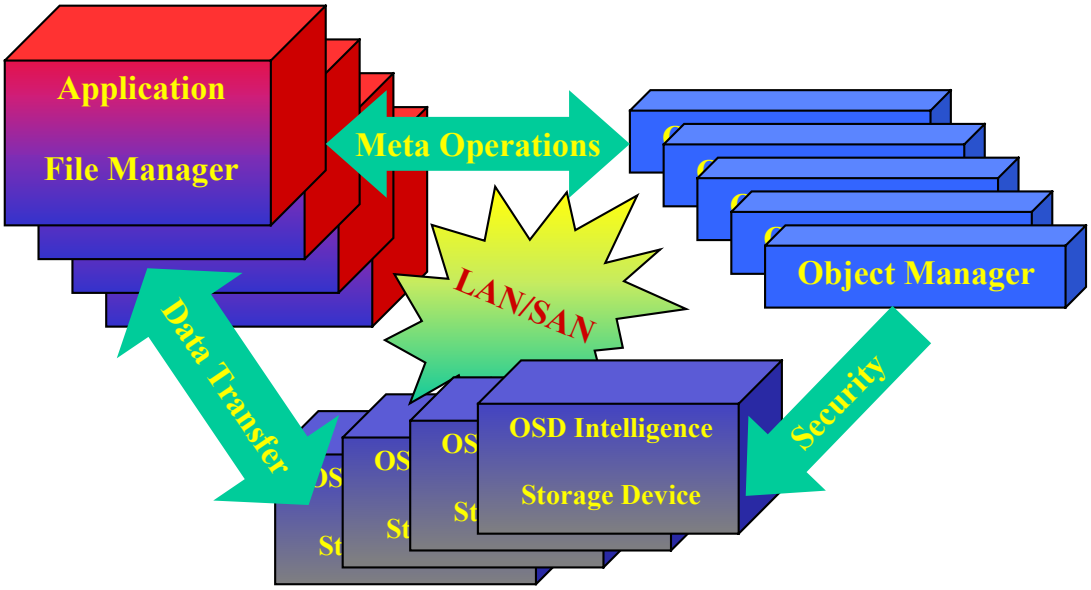


Diagram 5. Scaling a basic OSD architecture allows for increasing the number of OSD indefinitely as well as the application systems (clients). The Object manager can scale from a single system into a fully distributed cluster to accommodate the OSD and application system scaling. The transport for all these components can be either a LAN or SAN.

Basic OSD Architecture

One of the many motivations behind OSD was to take the strengths of each of the DAS/SAN/NAS architectures and incorporate them into a single framework. The basic OSD architecture and its scalability are shown in diagrams 4 and 5. There are many similarities between and OSD architecture and the DAS/SAN/NAS architectures. These include the use of Fibre Channel, Ethernet, TCP/IP, and SCSI protocols as transports and protocols. There are also several significant differences between OSD and the DAS/SAN/NAS architectures. These differences include the use of the following logical components:

- Object Manager
- OSD Intelligence
- File Manager

The Object Manager is used as a global resource to find the location of objects, mitigate secure access to these objects, and to assist in basic OSD management functions. This can be a single OSD that assumes these functions or it can be a completely separate, fully redundant cluster of systems. An Object Management Cluster would allow for scalability

in the number of objects that can be managed as well as the access performance of the Object Manager itself. It is important to note that the Object manager does not contain any user data or object meta-data nor does any of the data during a data transfer operation move through the Object Manager. The Object Manager is strictly used to facilitate location and secure access of objects.

The OSD Intelligence is the software (firmware) that runs on the storage device. It is responsible for interpreting the various OSD methods (commands): Create Object, Delete Object, Read Object, Write Object, and Get/Set Attributes. Furthermore, the OSD Intelligence can also provide the following capabilities:

- Object attribute interpretation
    - Object structure and relationship awareness
    - Object content awareness
    - Quality of Service (QoS)
    - Access Patterns
    - Security
- Sense of time
- Awareness and ability to communicate with other OSDs
- Device and data management

The OSD intelligence facilitates the communication of the OSD to the Object Manager for security purposes but mainly manages data processing and transfers between itself and the File Manager on the client requesting the data transfer. Since the OSD now has the intelligence to perform basic data management functions (such as space allocation, free space management, …etc.) those functions can be moved from the File SYSTEM manager to the OSD. The File SYSTEM manager now becomes simply a File Manager: an abstraction layer between the user application and the OSD. The File Manager provides backward compatible API for legacy codes to access files on OSD and, more importantly, it provides the security mechanisms required to ensure data privacy and integrity. More advanced capabilities of OSD can be exposed through the File Manager for user and system programs that wish to use them.

DAS/SAN/NAS/OSD Comparison

There are not actually any "new" data management functions in the OSD model. Rather it is simply a rearrangement of the existing functions in a general sense. From the user application point of view, the application creates, reads, writes, and deletes files as it always has. It does not know where the data is stored nor should it care. It does have certain data requirements (storage management, security, reliability, availability, performance, …etc.) that must be met and OSD provides a mechanism to specify and meet these requirements far more effectively than DAS/SAN/NAS. The following sections compare and contrast DAS/SAN/NAS to OSD in terms of the requirements listed in Table 1.

Storage Management[1]

Current estimates show that the cost of managing storage resources is about seven times the cost of the actual hardware over the operational life of the storage subsystems. This is independent of the type of storage (i.e. DAS/SAN/NAS). Given the tremendous growth in storage systems, storage resource management has been identified as the single most important problem to address in the coming decade. The DAS and SAN architectures rely on external storage resource management that is not always entirely effective and is in now way any kind of a standard. The NAS model has some management built into it but it too suffers from a lack of standards. The OSD management model relies on self-managed, policy driven storage devices that can be centrally managed and locally administered. What this means is that the high-level management functions can come from a central location and the execution of the management functions (i.e. backup, restore, mirror, …etc.) can be carried out locally by each of the OSDs and on an OSD peer-to-peer basis (i.e. a disk OSD backing itself up to a tape library OSD).

The DAS architecture is very simple to manage if there is only one system involved with some number of storage devices attached to it. All the management functions can be done from the one system that these devices are attached. However, if there is more than one system with storage devices attached, then it becomes increasingly difficult to manage all the storage devices because the management is distributed among all the systems that the storage devices are attached to. There is no central point of management in this case.

This problem is solved to some extent in a SAN configuration because ideally any one of the systems has access to all of the storage devices and management can be centralized on any one of these systems. A similar argument can be made for NAS devices since the network is a LAN and presumably any system on the LAN can see all of the NAS devices and hence can manage them all from a single system. Furthermore, the NAS devices have more "intelligence" built into them by their very nature (i.e. there is an OS with a file system, a communications stack, …etc.). This extra intelligence lends itself to the idea of self-managed storage making the overall task of managing storage resources somewhat easier. But is there a limit to the size of a system or the granularity of performance that can be managed in the NAS architecture?

The point here is that *centralized management* of storage resources (devices, space, performance, …etc.) with distributed administrative capabilities (i.e. the ability to carry out management functions locally) is essential to future storage architectures. In order to achieve this, the OSD architecture is designed to be self-managed thus more fully utilizing the OSD Intelligence built into each OSD. The devices will know how to manage each of several resources individually or through an aggregation of OSDs. These resources include (but not limited to):

- Space they have available at any given time
- Bandwidth has been requested
- Latency requirements of outstanding sessions

[1] In this section, the term "management" refers to the ability to install, configure, monitor, and administer the physical and logical storage devices as well as the space on these devices.

- The number of operations it is capable of performing in a given amount of time

Finally, OSD defines the concept of "object aggregation" whereby a hierarchy of OSDs can be made to appear as a single larger OSD. The resource management of this large aggregated OSD is done either through a single OSD at the top of the aggregation or can be done to each of the individual OSD devices in order to achieve maximum resource management flexibility.

Security

Security is second only to management in importance with respect to a data storage system. There are two basic threats that a secure system must guard against: External and Internal threats. External threats are attacks that come from outside the data storage system and outside the machines that are allowed access to the data on the storage subsystem. Internal threats are either benign or intentional. Benign threats are accidental access, modification, or corruption of data on a storage system. Intentional threats are intended to cause problems. In any case, multiple levels of security are necessary to authenticate, authorize access, ensure data integrity, and enforce data privacy.

Data security is becoming increasingly complex as the deployed systems and associated data storage systems grow in number and complexity. On the complexity scale, a DAS system is only as secure as the system that it is connected to. Assuming that the system is 100% secure, then access to the DAS device is very restricted.

By putting storage devices on a SAN however, there are more opportunities for access to the storage devices through other hosts that share the SAN. Generally, SANs are isolated and connected only to "trusted" host systems but there are still many other opportunities to connect to a SAN (i.e. through unused ports on a switch) and breach security. Since the SAN storage devices themselves do not have any notion of restricted access it is up to the host systems and SAN network infrastructure to enforce secure access to the storage devices.

NAS devices also have only as much security as the networks they are on and the firewalls and other security measures they implement. Because NAS devices tend to be on LANs the access restrictions may not be as stringent as those on SANs. However, since the NAS devices have some intelligence, they can implement more effect security measures than SAN devices.

The OSD concept incorporates a security model that includes four security levels:

- Authentication – you are who you say you are
- Authorization – you have permission to access to an object
- Data integrity – data is not modified or corrupted
- Data privacy – data is not to be seen by anyone else

The authentication is performed by the OSD transport layer. For example, for OSD over iSCSI over Ethernet, IPSEC would perform authentication. The remaining three levels are performed by the OSD itself. The authorization security mechanism is capability-based whereby the OSD manager gives capabilities to the clients and the clients present

these capabilities to the OSD. Finally, data integrity and data privacy are achieved through the use of cryptography. These are all features that make OSD security different from NAS security and certainly better than DAS and SAN security.

Device and Data Sharing

Concurrent device and data sharing is nonexistent on DAS systems unless the data is exported through an NFS or CIFS share to other systems. At that point the system essentially becomes a NAS device. Again, a SAN partially solves the problem by allowing any system connected to the SAN to access any device connected to the SAN. This is ideal for device sharing because the SAN provides a very high performance connection between any system and any device on the SAN. However, the problem of *data sharing* is left to the file systems to figure out. There are several ways to solve the problem of data sharing on a SAN, each with its own strengths and weaknesses. It is beyond the scope of this paper to describe these other than to say that data sharing is not always optimal on a SAN particularly in heterogeneous system environments (i.e. NT/Windows versus UNIX-based systems).

NAS devices are very good at sharing data even in heterogeneous system environments. The problem that NAS devices run into in this area is performance. There is a significant amount of overhead involved in performing each data transfer between the requesting system and the storage device where the bits reside. Furthermore, the store-and-forward model used by virtually all NAS devices can become a problem if not used correctly.

In the OSD model, the protocol is system agnostic and therefore system heterogeneous by nature. Since the OSD *is* the storage device and the underlying protocol is supported on either a SAN (SCSI) or a LAN (iSCSI), device sharing becomes simple. Data sharing is accomplished as a result of this as well. The objects contained on an OSD are available to any system that has permission to access them. It is *interpretation* of the object that needs to be common among the systems that becomes important for effective data sharing. That interpretation is outside the scope of OSD but the ability to access the object is there.

Storage Performance

Performance requirements differ from application to application but they come down to three basic components that can be described as:

- Bandwidth – the number of bytes per second that can be transferred between the requesting system and the storage device
- Latency – the time from the receipt of a request until the first byte of data is received
- Transactions rate – how many transactions of a particular size can be processed each second

The performance of DAS can be managed fairly closely because there is only one system talking to the device at any given time. This system can therefore reorder the request queue to a DAS device to minimize latency, manage available bandwidth, and maximize the number of transactions per second.

Similarly, on a SAN, any given system is presumably one of many accessing a storage device at any given time. On an individual basis, any given system can realize the same performance as a DAS provided no other systems are using the target storage device or any other required resources (hubs, switch ports, …etc.). The device-sharing capability of SAN however, makes the task of managing the storage performance exceedingly difficult. This is because the storage devices cannot differentiate between access requests and thus cannot give preferential treatment to any single request or set of related requests. Therefore, the bandwidth, latency, and transaction rates are not manageable on a SAN without some knowledge of the requesting system or the data being accessed. Neither of these pieces of information is available to the device in a standard SAN configuration.

A NAS device can address some of these issues since it can know something about the files being accessed and the host requesting access. The practice of file "tagging" is used to identify certain performance characteristics of files when they are accessed. For example, if a high-definition video file is being read from a NAS device, it could know that is must transfer this file using 80MB/sec of 120MB/sec of available bandwidth on a specific network connection leaving the remaining 40MB/sec to transfer other files through that same network interface. This preferential treatment of requests has the effect of providing guaranteed bandwidth, latency, and/or transactions per second. But again, the tremendous overhead of NAS makes it difficult to compete with either DAS or SAN for raw performance in these three categories.

The OSD model is very performance conscious. It is designed to allow performance characteristics of objects to be an attribute of the object itself and independent of the OSD where it resides. If the high-definition video file given in the previous example were on an OSD, it would have an attribute that specified an 80MB/sec delivery rate as well as a certain quality of service (i.e. a consistent 80 MB/sec). Similarly, there could be different attributes for the same object that describe delivery performance for editing rather than playback. In editing-mode, the OSD may have to skip around to many different frames thus changing the way the OSD does caching and read-ahead. Similarly, for latency and transaction rates, an OSD can manage these more effectively than DAS and SAN because it has implicit and explicit knowledge of the objects it is managing. The NAS concept of "file-tagging" is generalized and extended in the OSD model to accommodate current applications as well as future unforeseen application performance and functionality requirements.

Scalability

The term scalability means many different things. Hence another term, *extensibility* will be used in this section to expand upon the term scalability. Many of the items listed under the heading of "extensibility" can be accomplished by NAS devices. It is a question of the degree at which a storage device is extensible that is important. The OSD model is a single open model, not a specific proprietary implementation that is intended to provide the fundamental architecture that can extend far into each of the extensibility dimensions yielding years of opportunity and growth of storage systems built on the OSD model.]

This is only a partial list of extensibility dimensions but it demonstrates the breadth of characteristics that the OSD model encompasses:

- Density – the number of bytes/IOPS/bandwidth per unit volume. OSD on individual storage devices can optimize these densities by abstracting the physical characteristics of the underlying storage medium and hardware to objects.
- Scalability – what does that word really mean?
    - Capacity: number of bytes, number of objects, number of files, …etc. OSD aggregation techniques will allow for hierarchical representations of more complex objects that consist of larger numbers of smaller objects.
    - Performance: Bandwidth, Transaction rate, Latency. OSD performance management can be used in conjunction with OSD aggregation techniques to more effectively scale each of these three performance metrics and maintain required QoS levels on a per-object basis.
    - Connectivity: number of disks, hosts, arrays, …etc. Since the OSD model requires self-managed devices and is transport agnostic the number of OSDs and hosts can grow to the size limits of the transport network.
    - Geographic: LAN, SAN, WAN, …etc. Again, since the OSD model is transport agnostic and since there is a security model built into the OSD architecture, the geographic scalability is not bounded.
    - Processing Power – Given that the OSD model promotes the development of Active Storage Device technology it is reasonable to consider scaling the processing power on an OSD to meet the requirements of the functions the Active Disk is expected to perform.
- Cost – address issues such as $/MB, $/sqft, $/IOP, $/MB/sec, TCO, …etc.
- Adaptability – to changing applications. Can the OSD be repurposed to different uses such as from a film editing station to mail serving?
- Capability – can add functionality for different applications. Can additional functionality be added to an OSD to increase its usefulness?
- Manageability – Can be managed as a system rather than just a box of storage devices – Aggregated OSD management? Hierarchical Storage management?
- Reliability – Connection integrity capabilities
- Availability – Fail-over capabilities between cooperating OSD devices. Can this scale from 2-way failover to N-way failover?
- Serviceability – Remote monitoring, remote servicing, hot-plug capability, genocidal sparing. When an OSD dies and a new one is put in it's place, how does it get "rebuilt"? How automated is the service process?
- Interoperability – Supported by many OS vendors, file system vendors, storage vendors, middleware vendors.
- Power – decrease the power per unit volume by relying on the policy-driven self management schemes to "power down" objects (i.e. move them to disks and spin those disks down).

The DAS and SAN devices run into significant problems with extending into many of these dimensions. Even though these systems are built from many of the same physical devices, it is the efficiency with which they can be used that is a true differentiator between DAS/SAN and NAS/OSD. As was previously mentioned in the Storage

Performance section, DAS/SAN devices have very good performance but cannot manage that performance effectively or efficiently. A NAS system has the potential to manage performance but suffers from other performance-related issues due to the file-level access protocols (NFS/CIFS) used with NAS subsystems. Many of these extensibility dimensions are "afterthoughts" and were never designed into the NAS model from the beginning.

On the other hand, it is these extensibility features that the OSD architecture is designed to exploit to allow vendors to build more application-specific storage-centric systems thereby allowing storage vendors to more easily differentiate their products to address application requirements. The OSD architecture was designed with extensibility in mind rather than as an afterthought.

How OSD Relates to File Systems – An example in Scalability

Current file system technologies that access disk drives directly are "block-based" in nature. These file systems are responsible for the management of all available disk blocks on the disk storage devices they manage. Hence, the "file system manager" is the program that runs on a computer system that manages all the data structures on a disk storage device that make up a "file system". The file system manager will perform file creation, data block allocation, tracking of which files occupy which data blocks, control of access to these files, file deletion, and management the list of free or unused data blocks. In performing these functions the file system manager examines and manipulates on-disk data structures such as information nodes (inodes) and directory trees.

The file system manager manages two basic types of data: "meta-data" and "user data". Meta-data constitutes the file system *structure* that ultimately contains the user data files. Therefore, the file system manager has the ability to understand the "structure" of the "file system" but not the contents of the user data contained in the file system. Also, from the point of view of the file system manager, a disk storage device is simply a sequential set of disk blocks where a disk block is typically 512 bytes. All the meta-data and user data is mapped into this sequential set of blocks. From the point of view of the storage device, it only knows how to access 512-byte blocks. The storage device has no concept of the structure of these blocks as it relates to the file system or the data contained within the blocks.

The problem with the model of a "block-based" file system is that it can be severely limited in scale. As the number of blocks in the file system grows the task of managing the location of all the files and associated user data blocks grows as well. In 2001 the 180GB disk drive was shipped that contained 360,000,000 disk blocks. Three of these disk drives would constitute over one billion blocks to manage. A terabyte-sized file system would be made up of two billion blocks and a 10-terabyte file system, which is not uncommon these days, would be 20 billion disk blocks.

The OSD model would move the management of these individual blocks to the devices themselves. The file system manager would then only need to manage objects – a far more manageable problem. The fact that a disk device has blocks is completely hidden

from anything outside the disk drive itself. In fact, it does not even have to be a "disk" drive. It could be a solid-state device, a MEMS device, or a quantum crystal device. It no longer matters to the file system manager as long as the device can store and retrieve "objects". Now the file system manager only needs to worry about managing 500,000 objects and the fact that they take up the equivalent of 30 trillion 512-byte blocks is no longer directly relevant.

Functionality

DAS and SAN devices do two things and only two things: they write data and the read data. This is the limit of their functionality. NAS devices can perform more complex tasks such as snapshot backups, hierarchical storage management, data replication, …etc. because the NAS devices know certain attributes of the files they manage. However, most NAS device protocols still lack the extensibility to know and more effectively act upon the data they store.

The OSD model extends beyond the simple attributes of a file and allows for application-specific attributes that can specify relationships to other objects to form structures or functional attributes that can instruct the OSD to perform some operation (i.e. compression, encryption, …etc) on an object. The OSD model is intended to be used with the concept of Active Disks [Acharya] or Active Storage Devices. These devices can have significantly greater functionality than a simple DAS/SAN/NAS device because they can implicitly or explicitly act on the data they store.

It is this concept of Active Storage Devices that makes OSD so compelling for users *and* storage vendors. The reason for this is simple: users need to spend more time working on and with their data than trying to figure out how to manage it. Storage vendors need to have some way to significantly differentiate their storage products in an increasingly commoditized storage market. OSD provides and extensible mechanism to facilitate the incorporation of unique functionality storage devices thereby differentiating them from other storage products based on their *capabilities* not simply bandwidth, transaction rate, or capacity. Furthermore, since these storage devices are intelligent, they can be self-managed, autonomous "appliances" that are tailored to meet the requirements (processing, performance, reliability, …etc.) of *specific* applications.

OSD Roadmap

The concept of OSD has been around and in development for the past 10 years. Much of this work was pioneered by Garth Gibson and his research team at the Parallel Data Lab at CMU funded in part by Seagate. Recently however, an OSD Technical Working group has been formed as part of the Storage Networking Industry Association (SNIA – www.snia.org). The charter of this group is to work on issues related to the OSD command subset of the SCSI command set and to enable the construction, demonstration, and evaluation of OSD prototypes over the next several years. The command specification is to a point where working prototypes have been demonstrated by companies such as Seagate and Intel but no production or enterprise-level products have resulted from these prototypes yet.

Summary

OSD is an enabling technology for the development of active storage devices. By allowing the storage devices to understand, interpret, and act upon the data they store, new classes of storage-centric devices can be brought to market that enhance customer workflows while reducing total cost of ownership. OSD can also allow for more highly differentiated storage products based on capabilities rather than simple capacity, or raw performance thereby enhancing a storage vendor's ability to serve their respective markets.

References

[1] E. Riedel, G. Gibson, and C. Faloutsos, "Active Storage for Large-Scale Data Mining and Multimedia", *Proceedings of the 24th International Conference on Very Large Databases (VLDB'98),* August 1998

[2] K. Keeton, D. A. Patterson, J.. M. Hellerstein, "A case for Intelligent Disks (IDISKs)", *SIGMOD*, August 1998

[3] A. Acharya, M. Uysal, and J. Saltz, "Active Disks", *ASPLOS*, October 1998

[4] E. Riedel, G. Gibson, C. Faloutsos, G. Granger, D. Nagle, "Data Mining on an OLTP System (Nearly) for Free", *SIGMOD*, May 2000

[5] H. Gobioff, G. Gibson, and D. Tygar, "Security for Network Attached Storage Devices", *White Paper CMU-CS-97-185*, October 1997

[6] G. Gibson et al, "Filesystems for Network-Attached Secure Disks", *White Paper CMU-CS-97-118*, July 1997

[7] E. Borowsky et al, "Using Attribute-managed Storage to Achieve QoS", *Hewlett-Packard Laboratories White Paper*

[8] G. Gibson et al, "File Server Scaling weith Network-Attached Secure Disks", *SIGMETRICS '97*, June 1997

[9] E. Riedel (HP), G. Gibson, and C. Faloutsos, D. Nagle (CMU), Active Disks for Large-Scale Data Processing", *IEEE Computer,* June 2001

IP Storage: The Challenge Ahead

Prasenjit Sarkar, Kaladhar Voruganti
IBM Almaden Research Center
San Jose, CA 95120
{psarkar,kaladhar}@almaden.ibm.com
tel +1-408-927-1417
fax +1-408-927-3497

Abstract
Advanced networking technology has led to the genesis of the storage area network model, where host servers can access storage as a service from various devices connected to the network. While the initial approach to storage area networks has involved specialized networking technology, the emergence of Gigabit Ethernet technology has raised the question of whether we can use commodity IP networks for storage. This paper examines the issues involving IP storage networks and presents a performance analysis to dispel some of the myths and outline some of the challenges.

1 Introduction

With the steady increase in the storage needs of most organizations, block storage management is becoming an important storage management problem. Application servers, databases and file systems ultimately rely on the presence of an efficient and scalable block storage management system.

In the past, the storage model assumed the presence of storage attached to every host server. This type of host server-attached storage relied on the Small Computer System Interface (SCSI) protocol. The SCSI protocol emerged as the predominant one inside host servers due to its clean, well-standardized message-based interface. Moreover, in later years, it supported command queuing at the storage devices and allowed for overlapping commands. In particular, since the storage was local to the server, the preferred SCSI transport used was Parallel SCSI where multiple storage devices were connected to the host server using cable-based bus. However, as the need for storage and servers grew, the limitations of this technology became obvious. First, the use of parallel cables limits the number of storage devices and the distance of the storage devices from the host server. The limits imply that adding storage devices might mean the need to purchase a host server for attaching the storage. Second, the concept of attaching storage to every host server means that the storage had to be managed on a per-host server basis, a costly implication for centers with a large number of host servers. Finally, the technology does not allow for an easy sharing of storage between host servers, nor typically does the technology allow for easy addition or removal of storage without host server downtime.

The lack of scalability and manageability of the host server-attached storage model led to the evolution of the concept of a storage area network. Storage devices are assumed to be independent machines that provide storage service via a network to a multitude of host servers. The attraction of this approach is that host servers can share a pool of storage devices leading to easier storage administration. The advent of networking infrastructure capable of gigabit speeds further facilitates the service of storage over the network.

Furthermore, storage can be added, removed or upgraded without causing any host server downtime. In addition, the distance limitation of the host server-attached storage model is also removed.

Approaches to storage area networks have involved specialized technology such as HIPPI, VaxClusters, Fibre Channel and Infiniband [3][6][7]. The motivation behind the design is to construct a network that meets all the performance and connectivity requirements of a storage area network. The downside to these storage area networks is the requirement to purchase specialized adapters, switches and wiring for equipping the network. Furthermore, since storage area networks are not expected to be very high-volume, the cost of these components tends to be on the higher side in comparison to commodity Ethernet networks. Finally, all these specialized networks have very limited support for wide area networking and security. In fact, accessing such specialized storage area networks over long distances requires an IP network bridge.

The question then arises – is it possible to transport the SCSI storage protocol over commodity Ethernet IP networks [2] and still satisfy the performance requirements of storage area networks?

The advantages of IP networks are obvious. The presence of well tested and established protocols such as TCP/IP allow IP networks both wide-area connectivity as well as proven bandwidth sharing capabilities. Furthermore, the emergence of Gigabit Ethernet and the future arrival of 10 Gigabit Ethernet seems to indicate that the bandwidth requirements of serving storage over a network should not be an issue [1]. Finally, the commodity availability of IP networking infrastructure indicates the cost of building a storage area network will not be prohibitive.

This paper examines the issues involved in developing a high performance storage area networking solution. We present a performance analysis of a software-based IP Storage Area network. First, we measure the latency of block transfers to show that the protocol overhead of TCP/IP is minimal. Second, we do throughput measurements to show that while it is theoretically possible to saturate a Gigabit Ethernet network but that the CPU utilization is high compared to that in specialized storage area networks. We conclude this paper with an assessment of various hardware and software techniques that can help obtain high bandwidth at low CPU utilizations.

2 IP Storage

With the steady increase in the storage needs of most organizations, block storage management is becoming an important storage management problem. Both databases as well as file systems ultimately rely on the presence of an efficient and scalable block storage management system. The Small Computer System Interface (SCSI), rather than Advanced Technology Attachment (ATA), is the block management protocol of choice for most storage area network solutions because it supports command queuing at the storage devices and allows for overlapping commands. The SCSI protocol is mostly implemented over the parallel SCSI cable technology where multiple storage devices are connected to a SCSI bus via a cable. Though parallel SCSI technology supports gigabit

network speeds, the distance (few meters) and the connectivity limitations (16 devices to a channel) are hampering its acceptance as the gigabit networking transport layer of choice for the emerging large storage area networks. In addition, the parallel SCSI technology is more suited to attach to a specific host rather than being available as a network service which can be managed separately. Thus, specialized networking protocols such as Fibre Channel [3] and Infiniband [5] have been developed to overcome these limitations while still providing network-attached block storage at gigabit speeds.

The Fibre Channel protocol covers the physical, link, network and transport layers of the OSI network stack. Fibre Channel provides support for many different service classes. The Fibre Channel protocol contains a SCSI over Fibre Channel definition called FCP. The FCP protocol optimizes data transfer by enabling zero-copy transfers to the receiving host and reduces buffering requirements by making every frame self-describing. The FCP protocol also contains a simple and conservative flow control mechanism.

The Infiniband protocol also covers the physical, link, network and transport layers of the OSI network stack. The Infinband protocol provides support for many different service classes like Fibre Channel. In addition, the Infiniband protocol provides the QueuePair programming abstraction that allows application programs to transfer data directly from the network card into the application. The protocol provides the notion of verbs that allows application programs to send and receive data. The Infiniband protocol is similar to Fibre Channel in that it also supports a simple and conservative flow control mechanism.

Storage over IP is currently driven primarily by the iSCSI protocol [4] that defines the operation of SCSI over TCP and tries to leverage the existing TCP over IP over Gigabit Ethernet infrastructure. The goal of iSCSI is to leverage TCP flow control, congestion control, segmentation mechanisms, and build upon the IP addressing and discovery mechanisms to create a seamless and scalable storage area network. iSCSI can be implemented as a combination of network adapter card with the TCP/IP and iSCSI layers in software. This approach has the appeal of benefiting from the commodity appeal of existing network adapters and switches, an important factor in lowering infrastructure costs.

The challenges of building a storage area network over IP are not trivial. Detractors of IP storage area networks point out that the overhead of using TCP is prohibitive enough to result in poor latency for transaction-oriented benchmarks. It is also pointed out that common network application programming interfaces such as sockets do not allow for zero-copy transmits and receives of data leading to the overhead of multiple data copying [5]. Such data copying is considered harmful for overall throughput and will affect bulk-data scientific and video applications. Finally, data is transferred from the network adapter to the host machine using frame-size transfers. This means that every bulk data transfer may involve multiple interrupts instead of at most one interrupt in the case of specialized storage networks. Consequently, the interrupt overhead can be the limiting factor in peak throughput if the storage device or host server CPU spends the majority of its cycles processing interrupts.

3 Performance Analysis

We present a performance evaluation of a software implementation of IP storage and point out the performance characteristics that meet the requirements of storage area networks and those that do not. Our test-bed aims to determine the latency and throughput characteristics of a host server connected to a storage device over a Gigabit Ethernet network.

We use the iSCSI protocol [4] to transfer SCSI blocks between the storage device and the host server. The iSCSI protocol is a standard for transporting SCSI blocks over TCP/IP and is expected to be an IETF standard by early 2002. The key features of the iSCSI protocol are:

- Explicit login with the option to negotiate features such as security
- Authentication using SRP and other optional algorithms
- Trunking using multiple TCP/IP connections between storage endpoints
- Digests using CRC-32C and other optional schemes
- Encryption using IPSEC based algorithms
- Framing for faster recovery at high gigabit speeds
- Scalable discovery mechanisms using SLP and other protocols

The storage device is a dual-733 MHz Pentium III with 128 MB of memory and running iSCSI server software on top of Linux 2.4.2. The host server is an 800 MHz Pentium III with 256 MB of memory and running iSCSI client software on top of Linux 2.2.19. The two entities are connected via a Gigabit Ethernet connection over an Alteon 180 switch. The Ethernet frame size used was the regular 1500 bytes and no Jumbo frames were used. In addition, TCP/IP zero copy optimizations were not used. Instead, we relied on the standard socket interface that meant that the TCP copy-and-checksum routines were performed on both the host server and the storage device.

The test application resided on the host server and read raw SCSI blocks off a SCSI volume exported by the storage device. Since we wanted to isolate the efficiency of the transport, the application always read the same block so as to ensure a cache-hit. Otherwise, a cache miss would involve the RAID subsystem of the storage device and make it difficult to analyze the results. Writes were not measured as they can be done using various means (immediate, unsolicited, solicited) and add unneeded complexity to the analysis.

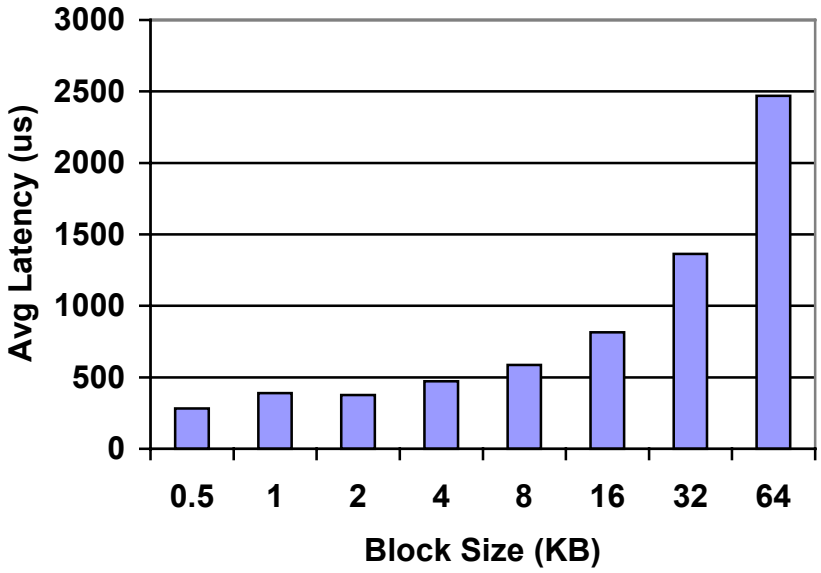
3.1 Latency Analysis

To measure latency, we used a single thread in the application to read raw SCSI blocks of various sizes from the storage device. For a particular block size, the same block was read 10,000 times and the average latency determined from the time required to perform the experiment. To measure throughput, we used 8 concurrent threads to read SCSI blocks of various sizes from the storage device. 8 threads were used because that is the concurrency limit imposed by the iSCSI client software in the host server. For a particular block size, each thread read a block 10,000 times and the throughput was calculated based on the time taken for all threads to finish reading the blocks. For the

throughput experiment, we measured the CPU utilizations of the host server and storage device using the *vmstat* utility.

The latency measurements shown in Figure 1 indicate a variation of average latency for 283 us for a 512-byte block to a high of 2469 us for a 64 KB block. The average latency values provide no meaning by themselves but are comparable (within 5%) of latency numbers obtained from the specification sheet of a Fibre Channel storage device for all block sizes [8]. We had expected the cost of TCP/IP segmentation to have an adverse effect on latency for the larger block sizes, but it appears that the Gigabit Ethernet adapter is doing a reasonable job of interrupt coalescing. This indicates that the TCP/IP fast path for transmits and receives does not impose a prohibitive overhead on latency. Consequently, we do not expect IP storage (even in its software incarnation with no optimizations) to have an adverse effect of transaction-oriented applications and benchmarks.

Figure 1. Latency Measurements



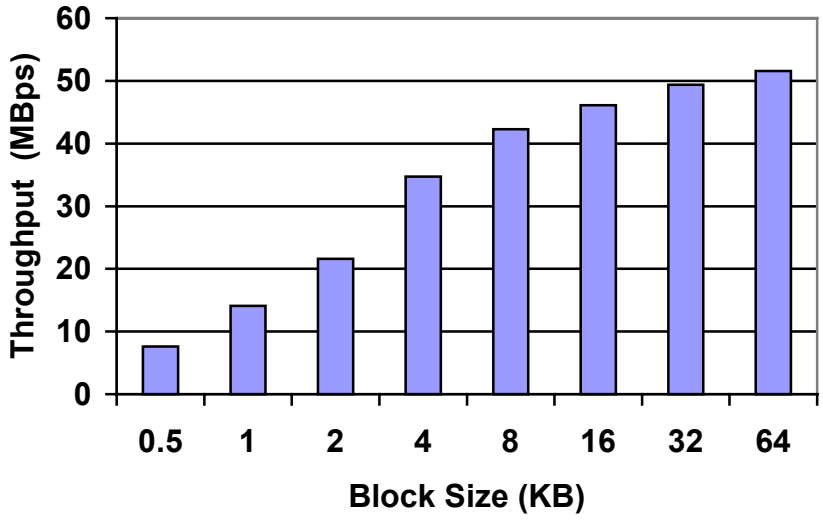
3.2 Throughout Analysis

However, the throughput measurements indicate a different story. Figure 2 indicates that while the average throughput from the storage device is competitive for the lower block sizes in comparison to that obtained from a Fibre Channel storage device, the peak throughput is about 60% less than what is obtainable from a Fibre Channel storage device[8]. In these experiments, the peak throughput is about 52 MBps for the 64 KB block size and is constrained by the CPU of the host server whose utilization is at 100%. A profiling of the CPU utilization of the host server indicated that the primary components were interrupt overhead (72%) and TCP copy-and-checksum (23%).

In addition, during the throughput experiments for the 64 KB block size, the CPU utilization of the storage device is at 51% indicating that the storage device is capable of delivering additional throughput. In fact, by using multiple initiators, we are able to obtain a throughput of 100 MBps at around 98% CPU utilization in the storage device. At this throughput, the constraining factor was the limit imposed by the network adapter. The CPU utilization figures were not available for the Fibre Channel storage device [8].

The CPU utilization of the host server is greater than that of the storage device because the host server is the receiver of bulk data. The receiving of data involves interrupting the host server every time a frame arrives and increases the interrupt overhead even if interrupt coalescing is used. This implies that if the experiments above involved writes, then the CPU utilization of the storage device would be higher.

Figure 2. Throughput Measurements



The results indicate that the main performance bottleneck in meeting the requirements of storage area networks is the high CPU utilization involved with bulk data transfers. The two main components of the high CPU utilization are:

- Interrupt overhead due to frame size transfers from the adapter to the host at high rates.
- The overhead due to TCP copy-and-checksum in standard TCP/IP stacks for bulk data.

4 Improvement Techniques

There are four potential avenues to reduce the high CPU utilization issues in an IP storage subsystem.

First, the interrupt overhead can be reduced by using 9KB Jumbo Ethernet frames, because this reduces the number of interrupts per bulk data transfer. For example, transferring a 32 KB data payload using the standard Ethernet frame may involve as many as 22 interrupts in the worst case whereas using the 9KB Jumbo Ethernet frame only 4 interrupts may be involved. However, the Jumbo Ethernet frames are not standardized and are not likely to be present in 10 Gigabit Ethernet.

Second, modified TCP/IP stacks with zero-copy transmit capability can be used to reduce the TCP copy-and-checksum overhead; the responsibility of generating the checksum is off-loaded to the network adapter. However, zero-copy receives are not possible on such stacks because the network adapters are typically unaware of the final destination of any frame.

Third, network adapters with TCP/IP offload engines (TOE) have been released [9] where the entire TCP/IP stack is offloaded onto the network adapter. This also reduces the TCP copy-and-checksum overhead. However, zero-copy receives are not possible on such stacks because the TCP/IP stack is also typically unaware of the final destination of any TCP/IP packet. There is proposed work to add enough application hints to the TCP/IP header to make zero-copy receives possible.

The fourth and most promising approach is the anticipated emergence of specialized adapters that have an iSCSI interface. This approach will reduce the interrupt overhead, as the iSCSI adapter will ensure at most one interrupt per data transfer. In addition, offloading the protocol processing to the adapter will eliminate TCP/IP copy-and-checksum overhead. The disadvantage of this approach is that the use of such specialized adapters implies that commodity network adapters cannot be used in IP storage area networks. However, one can still use the existing switches and wiring present in commodity Ethernet networks.

5 Conclusions

Advanced networking technology has led to the concept of storage networks where pooled storage is available as a service to host servers. The emergence of Gigabit Ethernet technology has raised the question of whether we can use commodity IP networks for storage instead of specialized storage area networks. This paper examines the issues involving IP storage networks and presents a performance analysis focusing on latency and throughput. The results indicate that the main performance bottleneck in meeting the requirements of storage area networks is the high CPU utilization involved with bulk data transfers. The two main components of the high CPU utilization are the interrupt overhead due to the bulk data transfers as well as the TCP copy-and-checksum overhead. We finally present four potential avenues to reduce the high CPU utilization issues in an IP storage subsystem.

References

[1] A. Gallatin, J. Chase, and K. Yocum, "Trapeze/IP: TCP/IP at Near-Gigabit Speeds", *Proceedings of USENIX Technical Conference (FreeNix Track)*, June 1999.
[2] R. Van Meter, G. Finn and Steve Hotz, "VISA: Netstation's Virtual Internet SCSI Adapter", *ASPLOS-VIII*, October 1998.
[3] A. Benner*, "Fibre Channel: Gigabit Communications and I/O For Computer Networks"*, McGraw-Hill, 1996.
[4] J. Satran et al., "iSCSI", IETF Work in Progress (IPS group), http://www.ietf.org/html.charters/ips-charter.html, 2001.
[5] Hsiao Keng, and J. Chu, "Zero-copy TCP in Solaris", *Proceedings of the USENIX 1996 Annual Technical Conference*, January 1996.
[6] http://www.infinibandta.org
[7] K.Voruganti, and P. Sarkar, "An Analysis of Three Gigabit Networking Protocols for Storage Area Networks'. *20th IEEE International Performance, Computing, and Communications Conference"*, April 2001.
[8] Mylex Corp., "White Paper on the Performance of the Mylex SanArray Pro FF2 Storage Controller", Mylex Technical Report, 2001.
[9] http://www.alacritech.com

File Virtualization with DirectNFS

Anupam Bhide, Anu Engineer, Anshuman Kanetkar, Aditya Kini
{anupam, anu, anshuman, aditya}@calsoftinc.com
CalSoft Private Limited, Pune 411 013, India
Tel: +91 20 567-4644
Fax: +91 20 567-7279
Christos Karamanolis, Dan Muntz, Zheng Zhang
{christos,dmuntz,zzhang,gary_thunquest}@hpl.hp.com
HP Research Labs
1501 Page Mill Road, Palo Alto CA 94304-1126
tel: +1 650 857-1501
Gary Thunquest
HP Colorado
{gary_thunquest}@hp.com

Abstract

There is a definite trend in the enterprise storage industry to move from Network Attached Storage (NAS) solutions to high performance Storage Area Networks (SAN). This transition is not easy because of the well-entrenched NAS infrastructure that has already been deployed. This paper attempts to define a file system that can leverage the existing NAS software infrastructure along with evolving SAN technology to provide the benefits of high performance storage access while reducing the cost of migrating to these networks.

In this paper, we propose a new network file system, DirectNFS, which allows NAS clients to take full advantage of the performance and scalability benefits of SANs. In order to achieve this goal, the system presents a NAS interface to existing NAS clients while allowing DirectNFS clients to access storage directly over shared SAN, i.e. clients bypass the server for data access. A server maintains the NAS interface for legacy clients and arbitrates access to metadata by DirectNFS (SAN aware) clients. This metadata server ensures that the system is operable for both legacy NAS clients as well as DirectNFS clients. The communication protocol of DirectNFS is designed as an extension of traditional network file systems protocols, such as NFS and CIFS.

A prototype of DirectNFS has been built for Linux, as an extension to the native NFSv2 implementation. Initial results demonstrate that the performance of data intensive operations such as read and write is comparable to that of local file systems, such as ext2.

1. Introduction

For the past few years, there has been an increasing trend to replace NAS storage systems by SAN. The primary reasons for this migration have been the increased data storage requirements that constantly plague the enterprise computing environment. SANs provide seamless expansion, combined with high throughput, and increased manageability. However, NAS architecture has been around for many years and has a well-entrenched installed base. The migration to SAN makes this NAS infrastructure obsolete and adds to the cost of already expensive SAN systems. One major drawback of the SAN systems that are deployed now is the lack of interoperability. However, this

situation will eventually be remedied as more users adopt SANs and as SAN standards evolve.

Today, with multiple operating systems and multiple vendor platforms present in most data centers, SAN inter-operability is highly valued. NAS technologies, on the other hand, are mature and interoperable. They use de-facto standards such as NFS[1] and CIFS[2] to provide data access. NFS clients are available for almost all platforms. Both NFS and CIFS have mechanisms to control and synchronize simultaneous access to shared data. These inherent features of NAS were taken advantage of in the design of DirectNFS.

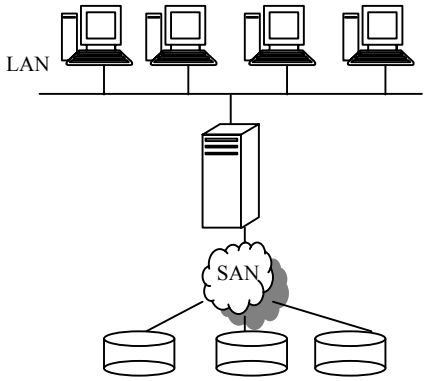


Figure 1: SAN with NAS Clients

A simple way of using the SAN, as shown in Figure 1, is to retain the familiar client/server model, with all the storage resources on the SAN appearing as local disks to the server. All the file accesses by clients in this scenario are forced to pass through the file server. This creates heavy loads on the file server.

In order to eliminate this overhead of data being copied through both SAN and LAN, the clients must be given the ability to access the data directly through SAN. To enable clients to access data directly, we have to provide them with a file location map that describes on which device and on which block the file data resides - information that is maintained as part of the metadata of the file system.

There have been different solutions to the distributed storage problem, ranging from "Shared Everything" to "Shared File Volume" architectures. In a "Shared Everything" filesystem, all clients maintain data as well as metadata portions of the file system. Most of the cluster file systems follow this approach (Petal /Frangipani[3], GFS[4]). In a "Shared File Volume" filesystem, one central entity is in charge of updating the data and metadata. Most client / server file systems follow this approach (NFS, CIFS). In a "Shared Everything" approach the implementation of the file system and its recovery on failure is complex. On the other hand, in a "Shared File Volume" approach, the scalability and performance of the file system are limited due to the existence of a single server. In the design of DirectNFS we have chosen to tread a middle ground between these two approaches. We have chosen to create a shared architecture for data, by making the clients aware of the physical layout of each file, which allows the clients to access data directly through the SAN. However, we do not allow clients to modify the metadata directly. Once we allow the clients to access data directly, the NAS-provided guarantees of single system semantics break down. This is unacceptable because a lack of single system semantics would lead to corruption of the file system. The solution is to create an entity that enforces these semantics, and this entity in DirectNFS is known as the metadata server. The metadata server is responsible for all metadata modifications in the file system. Since most filesystem metadata operations are atomic in nature, a single authority in charge of metadata modifications makes file system implementation and recovery easier. The metadata server also provides NAS

interfaces to legacy clients for interoperability. This approach does have a drawback of introducing a single point of failure (metadata server) which makes the system less fault tolerant as compared to "shared everything" file systems. We believe that the potential gains from implementing a "shared everything" file system and making it compatible with legacy clients are not worth the complexity of the implementation.

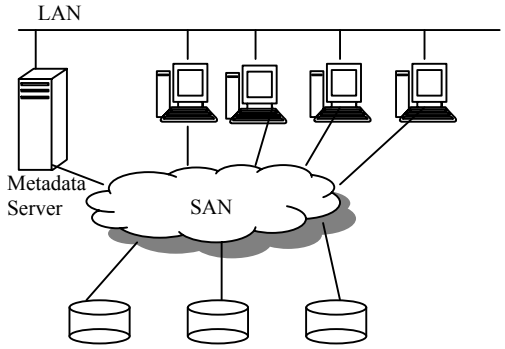


Figure 2: DirectNFS Network Architecture

DirectNFS clients are allowed to cache the block metadata, or the information pertaining to location of files. Coherency is enforced using a lease protocol. The metadata server acts as an arbitrator between the clients to make sure that the cached metadata is valid. The network architecture of DirectNFS is shown in Figure 2. By adding a SAN connection and DirectNFS software to each client, clients can utilize the file server for file system metadata access, locking, and coherency, but they read and write file data directly from the storage, bypassing the file server. The introduction of a simultaneous data access path can improve file serving performance through parallel and direct transfer of data between the data sources and the client systems. This also achieves better utilization of the file server by reducing the CPU and network load on the metadata server. Clients that either do not have a SAN connection or do not have the DirectNFS software can continue to access data through the server using the NFS or CIFS protocol clients, which they already have. This makes DirectNFS a powerful tool in migration of existing LAN/NAS combination to SAN.

We have implemented a GNU/Linux prototype of DirectNFS. Many platforms such as FreeBSD, Solaris and HP-UX were considered for reference implementation. GNU/Linux was chosen primarily because of the ease of source code availability, general acceptance in terms of usage and the support from the large community of hackers.

In our GNU/Linux prototype, we have demonstrated throughput comparable to that of a local (ext2) file system. Thus, we provide client applications the ability to have both shared file access and near local file system performance simultaneously. We have also observed lower server resource utilization in the metadata server compared to a NAS server, which implies that DirectNFS can support more clients than traditional NAS servers. DirectNFS implementation is transparent to applications running on the clients: no source code changes are necessary to client applications. During system operation, DirectNFS can be turned on or off without altering the file system semantics.

In this paper, in section two we talk about the goals associated with the DirectNFS design, section three talks about the design in detail. Section four of this paper deals with the Linux prototype. Section five discusses work done previously in this area. In section six, we highlight the performance achievements of DirectNFS. We present future directions for DirectNFS in section seven, and conclude in section eight.

2. DirectNFS Design Goals

In this section, we provide a list of design objectives of the DirectNFS architecture. In subsequent sections, we discuss the DirectNFS architecture in greater depth.

- *Storage Scalability* - Storage space must scale well with the continuous accumulation of data.
- *High Performance* - DirectNFS aims to provide a high performance remote file system, with orders of magnitude performance improvements over traditional NAS protocols.
- *File System Scalability and Recovery* - To create a simple distributed file system that can provide both scalability and recoverability.
- *Independence from Physical File Systems* - DirectNFS must be able to run irrespective of the underlying physical file system that is used for storage.
- *Portability* - DirectNFS should be portable to other Operating systems without much effort.
- *File Virtualization over SANs* - Enable the seamless integration of Storage Area Networks into NAS environments by adding a "File Virtualization" layer on top of the block-level interface that SANs provide.

3. Design

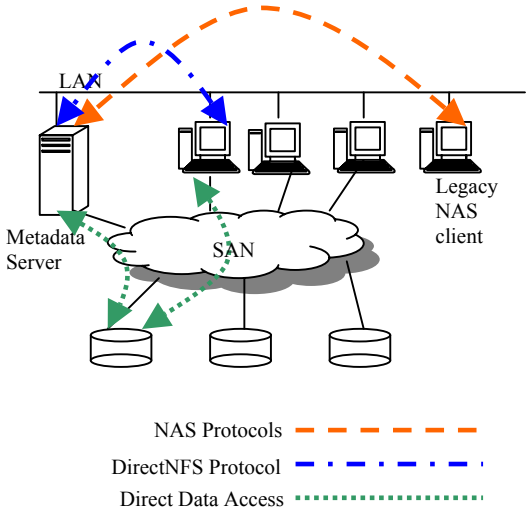


Figure 3: DirectNFS Architectural Overview

The basic philosophy behind the design of DirectNFS is the separation of data from metadata operations to increase parallelism in file system operations. Only read and write operations are taken over by DirectNFS client software, all the other file system operations are still performed through the NAS protocol. This makes DirectNFS design portable, thereby enabling us to use the same design on a host of other platforms including NT, BSD, Solaris and HP-UX.

The Figure 3 shows these operations more clearly, the communication between the DirectNFS client and metadata server. This communication includes lease protocol communication to maintain metadata coherency, the metadata information requests and NAS protocol functionality that is not intercepted by DirectNFS. The legacy NAS client communicates with the metadata server as if it were an ordinary NAS server.

3.1.Architecture Overview

This section provides an overview of DirectNFS architecture including DirectNFS extensions to the NFS protocol, cache coherency mechanisms, optimizations, and security.

3.1.1. Extensions to NFS

DirectNFS defines extensions to the NFS-RPC[5] protocol that implement the separation of the data/metadata path. This includes new RPCs used by the clients to retrieve the physical location of files on the storage (block lists) and additional RPCs to enforce cache coherency. The native RPC set of NFS is used to perform metadata operations on the server.

The new RPCs implemented by DirectNFS are,

- *GETBLKLIST* : This RPC allows the clients to get the block list of the files that are present in the system. The arguments to this RPC are the NFS file handle and the byte range for which the block list is requested.
- *GETLEASE* : This RPC is used by the DirectNFS client to acquire the lease for locally cached metadata. This RPC can be piggy backed on the GETBLKLIST RPC. The argument is the NFS file handle and duration. The reply sent by the server indicates whether the requested lease has been granted or denied.
- *VACATELEASE* : This RPC is used by the metadata server to ask a client to release the lease it has on certain file. The argument to this RPC is NFS file handle.
- *VACATEDLEASE* : This RPC is issued by the client, when it releases an lease due to the request from the metadata server.

Using these RPCs, clients are able to retrieve the physical locations of files and access them directly without conflict.

3.1.2. Metadata Caching and Cache Coherency

DirectNFS clients use extensions to the NAS RPC protocols to retrieve file metadata, i.e. physical block and device numbers. This file metadata is then cached locally on the client in a Block-Number Cache (BNC). This allows DirectNFS clients to cache the most frequently used physical block numbers for files that are most frequently used. However, introducing a distributed cache also introduces coherency issues, which we solve using a leases-based protocol.

A lease is a time-bound object granted by a lease server to a lease client. In DirectNFS, a lease is granted on a per-file basis to clients by the metadata server. The lease guarantees the client that as long as its lease is valid; it holds the most current copy of the data object (i.e. the cached list of blocks for the file). Multiple clients are allowed to share leases on the same data object for read-only access. However, any changes to this data by a third party can only be made when the server has revoked all other leases. This revocation is either done explicitly by notifying the client, or implicitly, if the leases time out. In either case, once the lease expires, the lease-holder has to discard the cached data protected by the lease.

The time-bound property of leases ensures simple recovery of clients/servers in case of a crash or network failure. Neither the client nor the server maintains any state. In case of a system crash, the leases that were issued before the system went down will expire, which brings the system to a known, stable state. This makes the recovery algorithm extremely simple to implement, especially when compared to the NLM protocol or other Distributed Lock Managers.

However, this coherency mechanism does not protect the system against SAN partitions, which may lead to data corruption – it is assumed that the SAN provides a reliable and available service for data delivery.

When the DirectNFS client needs to read/write a block of data, it first ensures that it has the right lease for the kind of access it needs to perform. The interaction between DirectNFS clients and metadata server for lease acquisition in write and read scenarios is illustrated in figures 4 and 5 respectively.

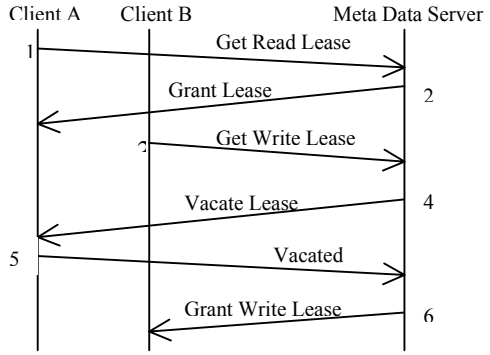


Figure 5: Sequence Diagram for Lease Protocol Interactions (Read-Write Conflict Case)

Once the lease has been validated, the client looks up the Block Number Cache for the physical location of the data. The metadata server is then queried for metadata information only in the event of a cache miss.

Metadata caching is augmented with "write allocation gathering". This is the process of deferring disk block allocations during file writes. In DirectNFS, we do write allocation by gathering write requests at the client. Smaller byte-range requests are merged into larger requests, thereby reducing the number of metadata requests to the server. This significantly improves performance, by reducing the number of requests to the server that the server has to service. "Write gathering" [6] performed by NFS is similar in its approach and it is used to exploit the fact that there are often several write requests for the same file presented to the server at about the same time.

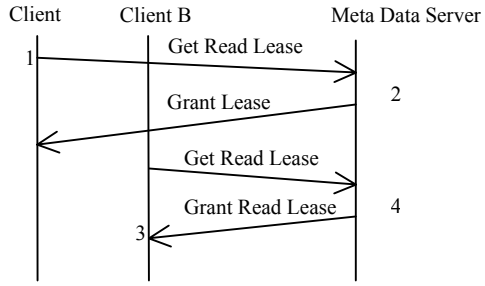


Figure 4: Sequence Diagram for Lease Protocol Interactions (Read-Sharing Case)

3.1.3. Write Gathering

Distributed-system file access patterns have been measured many times[7]. It has been found that sequential access is the most common access pattern.

Under DirectNFS, for every write request, a cache miss would result in a GETBLKLIST RPC being sent to the metadata server. To improve write performance, a technique called write gathering is employed that exploits the fact that there are often several write requests for the same file called about the same time. With this technique the data portions of these writes are combined and a single metadata update is done that applies to them all. In this way, the number of RPCs being sent out would dramatically reduce, and considerably improve write performance.

The performance for write gathering depends on the periodicity of the deferred write requests to the server. Two events can trigger this: the write back cache being flushing periodically and an eviction notice received at the client.

3.1.4. File Virtualization

One of the major issues of merging SAN and NAS is the basic unit upon which they operate. The legacy NAS protocols operate at a "File" level abstraction. However, the SAN systems normally present the block level interfaces that are leveraged by filesystems.

In the DirectNFS design, we were faced with the problem of maintaining support for legacy clients, which meant that we needed to maintain the file level abstraction. On the

other hand, the benefits of the SAN can be leveraged if and only if we went down to the block level. In order to solve this problem we created a "virtualized file interface over SAN", where the legacy NAS clients are under the impression that the NAS server stores the files, but the DirectNFS clients went below the file abstractions to leverage the SAN performance by using block device interface directly. In order to implement this duality, we had to achieve the data-metadata split and create other mechanisms like the lease framework in order to tackle complexities arising out of the merger of SAN into NAS.

The DirectNFS file system had to merge these two different worldviews to create a high performance distributed file system, which offered a NAS interface. This was achieved by maintaining a "Virtual File Interface". However, the DirectNFS client behavior can be compared more to block device driver, than really a NAS file system client. In other words, we introduced the SAN abstractions and performance to the NAS protocols without breaking it. This unification of SAN of under NAS is what is referred to as file virtualization in DirectNFS.

3.1.5. Security Considerations

There are certain assumptions that are critical to DirectNFS architecture that need to be pointed out while understanding the security mechanisms in DirectNFS. They are

- The base NFS protocol operates on atomic data entities known as files.
- DirectNFS does not alter the semantics of NFS protocol
- DirectNFS relies on the file system and block device layer to provide security that is needed.

DirectNFS has modified the VFS layer[8] of NFS communication not the NFS semantics. The real physical file system must be present for DirectNFS to work. This is a strict requirement because we still rely on the file abstraction to maintain the coherency of data.

In DirectNFS, the file system layer is responsible for security and data coherency. In order to solve the coherency problem at file level, we have created a framework of leases ensuring that coherency is maintained at the file system level.

However, in case of rogue agents who can access the storage system at the block interface by bypassing DirectNFS completely, the possibility of unauthorized access remains, unless the block access mechanism (block device driver) provides security. We currently provide only file level security but do not provide block level security. NASD [9] addresses the issue of block level security with the help of special hardware. If the shared storage contains security mechanisms, for example iSCSI [10] has security mechanisms built in and when DirectNFS operates on those environments it can be made to run in a secure mode by leveraging these underlying mechanisms. Thus DirectNFS relies on existing infrastructure to take care of security (iSCSI, Fiber channel[11], NFS). This is a conscious design decision made in favor of making this protocol run on extremely varied range of hardware.

4. Implementation of the Linux Prototype

The implementation philosophy of DirectNFS was to reuse existing libraries as far as possible and to maintain portability. It was implemented as a kernel loadable module on Linux 2.4.4, and it consists of roughly 8000 lines of code on the client and 1500 lines of code on the server.

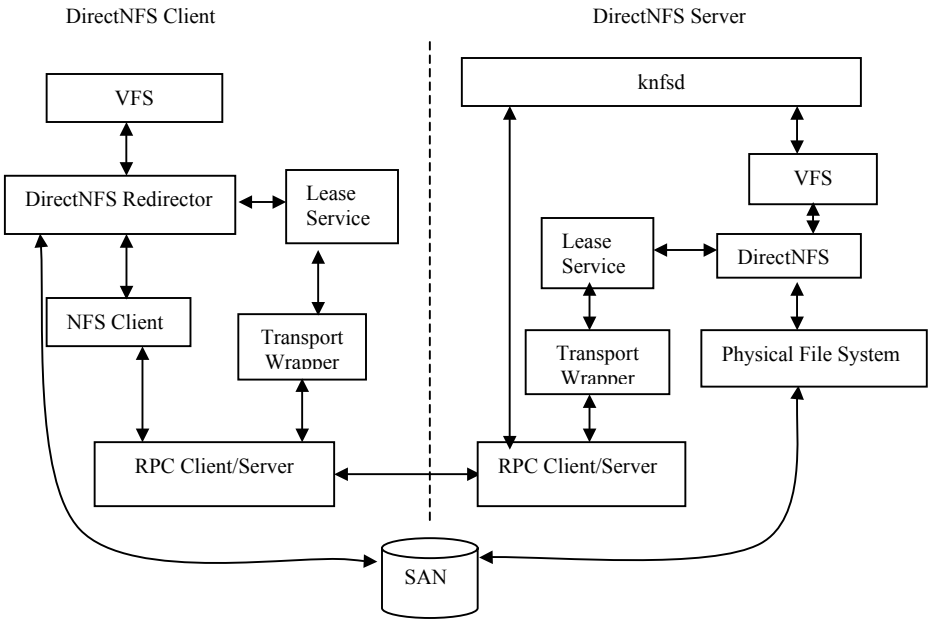


Figure 6: DirectNFS Software Architecture

4.1.DirectNFS with FiST

In order to make the implementation easier and portable we have used FiST (File System Translator). FiST [12] is a stackable file system generator. It defines its own highly abstract Domain-Specific Language (DSL) for describing file-system filters. A compiler translates the DSL description to C code for various operating systems. FiST also provides the necessary infrastructure for interposing the generated filter between the VFS (Virtual File System) and the natively installed file systems in the kernel. FiST played an important role in the initial phase of the implementation, when we used it to generate a code skeleton for a simple, pass-through file system that interposed itself between the VFS layer and the NFS client.

On the DirectNFS client, the Linux DirectNFS module can be thought of as consisting of these sub-modules:

1. **The DirectNFS Filter/Redirector** – This component interposes itself between the VFS and the NFS client module. It intercepts all file I/O operations (read, write) and redirects them as block I/O requests over the SAN. This was achieved by modifying the basic FiST-generated filter to enable us to intercept I/O operations instead of passing them down the file system stack, which is the default FiST policy. The I/O interception code in the redirector is system-dependent. The redirector also contains the Block Number Cache, where the client caches location information for each file that is accessed over DirectNFS.

2. **Leasing Service** – This is a distributed protocol, which allows multiple DirectNFS clients to keep their cached metadata coherent. The leasing service has been built as a library that is independent of the transport mechanism underneath it. This allows us to plug in any transport mechanism by writing a transport wrapper for the mechanism.
3. **Transport Wrapper** – This provides an interface between the leasing service and the transport layer, in this case - RPC. This wrapper allows the file system client to query file location information (i.e. block numbers) from a central server and to communicate lease requests to the server.

The DirectNFS server module consists of:

1. **Leasing Service** – This is the server-side counterpart of the leasing service. It is responsible for maintaining a list of lessees for each file, and to resolve lease conflicts.
2. **Transport Wrapper** – The transport wrapper on the server as on the client provides an interface between the leasing service and the transport layer. This wrapper allows the server to interface with file system clients that query for file location information and to communicate lease rejections or grants to them.
3. **DirectNFS client** – A DirectNFS client is interposed between VFS and the physical file system, to provide lease-based coherency for locally originating file accesses. This could be from local applications trying to access the physical file system or from knfsd while it is serving legacy NAS clients.

The DirectNFS module on the client is responsible for trapping file open, close, sync, unlink, read, and write calls. Since these operations access the location information of the file, the file's lease is tested for validity. If the lease is invalid, it is acquired by issuing a GETLEASE RPC to the metadata server. For read and write operations, the Block Number Cache is looked up for cached block numbers. On a cache miss, a request is sent to the server, with a piggybacked lease request, if required. This is done with the GETBLKLIST RPC. Once the client is granted a valid lease on the file, and receives the requisite file location information, it accesses those blocks directly over the SAN.

In the event that the client receives a VACATE RPC, which signals the server ordering an eviction of the lease that the client holds on the metadata, the client flushes the cache that is associated with the file, and then proceeds to inform the DirectNFS server by sending the VACATED RPC.

Note that the DirectNFS Leasing Service makes the following assumptions:

1. The lease is time-bound, has a fixed duration, and must be renewed explicitly at the server in order for its time period to be extended.
2. The clock skew between the participating entities in the lease protocol is bounded.
3. The time taken by the client to flush its cached after eviction is bounded.

Lease conflicts are resolved by the lease server using the matrix in Table 1.

	Read	Write
Read	Shareable	Non Shareable
Write	Non Shareable	Non Shareable

Table 1: Compatibility Matrix for DirectNFS Leases

5. Performance

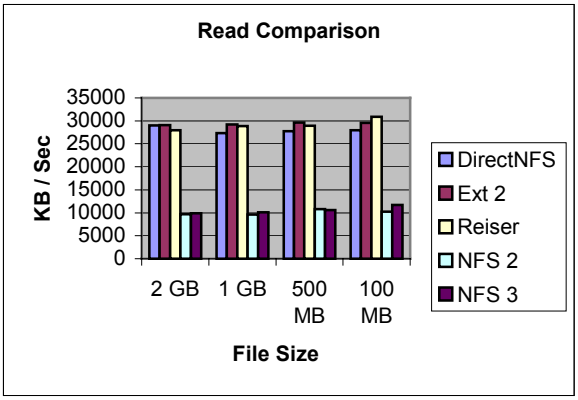


Figure 8: Read Comparison

One of the principal objectives of DirectNFS is performance. In this section, we present the performance numbers that we obtained from the prototype implementation. We have measured the performance of DirectNFS against other file systems like ReiserFS[13], ext2 and NFS versions 2 and 3[14]. The systems under test were three HP Netserver LC 2000, Pentium III's -933 Mhz with 128 MB RAM and 256KB L2 cache. The machines were running Redhat Linux, with custom-built kernels from the 2.4.x series. They were connected to a JBOD (HP Rack Storage/12) of four Ultra 3 Hot-Swap SCSI[15] disks 9 GB each. The system was set up in a SCSI multi-initiator arrangement, with two machines acting as DirectNFS clients, and one machine as the DirectNFS metadata server, with all three machines sharing access to the JBOD through a shared SCSI bus. This was used to emulate a SAN. The benchmarking utility that we used was Iozone [16].

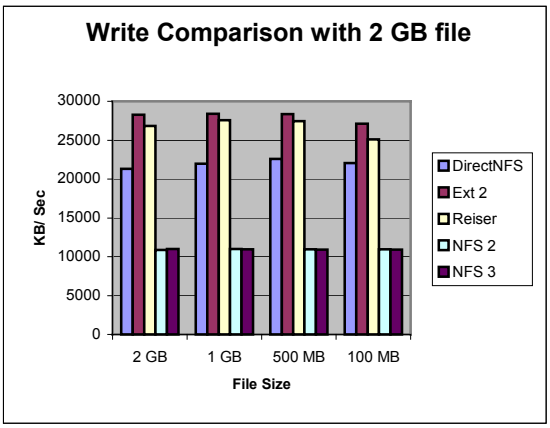


Figure 7: Write Comparison

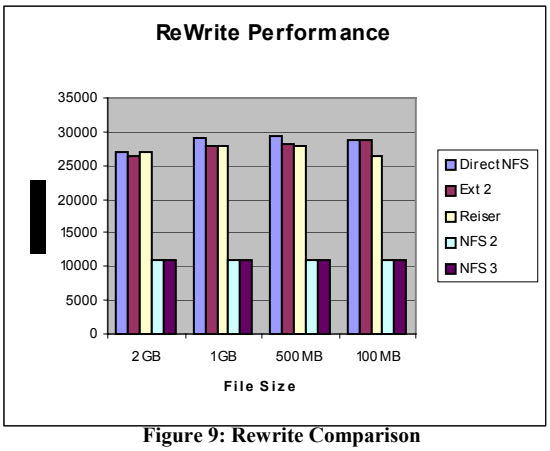


Figure 9: Rewrite Comparison

We benchmarked the performance of DirectNFS with varying file sizes and record sizes. From the data, we observed no significant variations in the comparative figures. Hence, we have included the performance figures of read, write, reread and rewrite of a 2GB file over ResierFS, DirectNFS, Ext2, NFS2 and NFS 3. Figure 7 is a the performance graph of various file system read throughputs for varying file sizes, with fixed record size of 256 KB. The rest of the graphs - Figures 8, 9 and 10 - carry comparisons of write, re-read and re-write operations. These figures indicate that DirectNFS performances are comparable to local file systems.

The write performance of DirectNFS shown in Figure 8 is slightly worse than Ext2 and ReiserFS. Re-read and re-write were tested so that we could measure the effects of the Linux page cache.

We have measured throughput for these four operations with varying file sizes starting from 100 MB up to 2GB and varying record sizes starting from 4 KB up to 256 KB. Since the throughput figures we obtain did not vary significantly across these series, we reproduce data for 256KB record sizes only. The file sizes selected were suitable large, as we expect the primary use of DirectNFS to be multimedia applications (e.g. streaming media servers), which use large files.

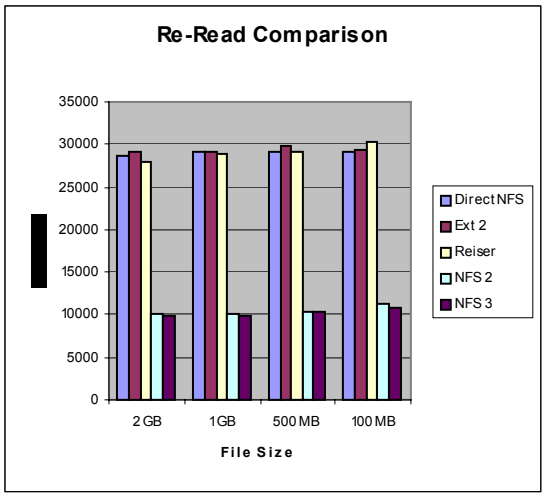


Figure 10: Re-Read Comparison

Note that NFS v2 and v3 throughput figures that we measured were very close to each other. Even though NFS 3 implements Asynchronous writes, NFS 2 clients under the Linux use write caching and by default run with synchronous writes set to off. This hides the RPC latency of NFS from client applications. However, we wanted to compare against real world performance and hence we tried to measure against the fastest NFS performance possible.

From a glance at the throughputs for read and re-write tests, it appears that DirectNFS performance comes close to matching the performance of both ReiserFS as well as ext2. This can be accounted for by the metadata cache, which contains logical to physical block translations, and improves the performance of DirectNFS, bringing it close to ext2 and in some cases surpassing it (this is because the mapping function for the cache is less expensive than the corresponding lookup operation in EXT2 or ReiserFS). We also examined the effect of record size on performance. Figure 11 is a comparative graph for the read operation for various file systems with fixed file size but with varying record size. We did not observe any significant effect of record size on throughput of any of the file systems under consideration. This is most likely due to the pre-fetching in the VFS layer.

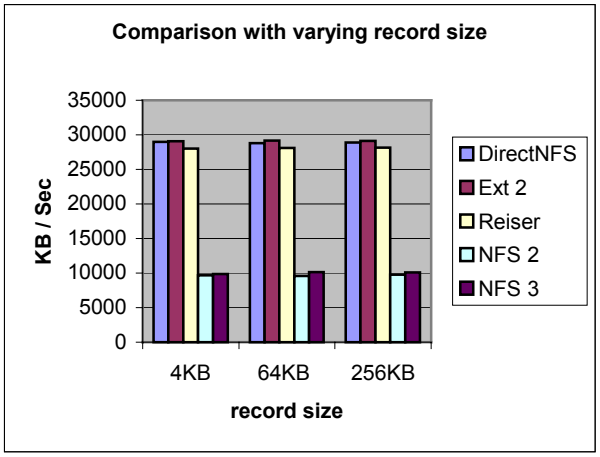


Figure 11: Comparison with varying record size

If we look closely at the performance relative to NFS2 or NFS3, we see that the performance improvements that are achieved are significant, and are 2 to 3 times that of the Linux implementation of NFS.

There are two measures of goodness for a network file system, the first is the throughput that each client can expect from the file system, and the second is the server scalability. DirectNFS

addresses both of them by increasing the client throughput by a factor of 2 to 3 as compared with competing NAS technologies like NFS, and increases the server scalability significantly by reducing CPU utilization at the server.

A look at Figure 12 shows the relative CPU utilization of DirectNFS with NFS. The tests that were carried out were sequential read, sequential reread, sequential write, and sequential rewrite. Now, if we look at the NFS performance, we can conclude that NFS (with a single client running Iozone tests on a file of size 1GB) requires a mean CPU utilization of more than 20%. Thus, the scalability of the server is limited to the number of clients that access the NFS server at any point of time.

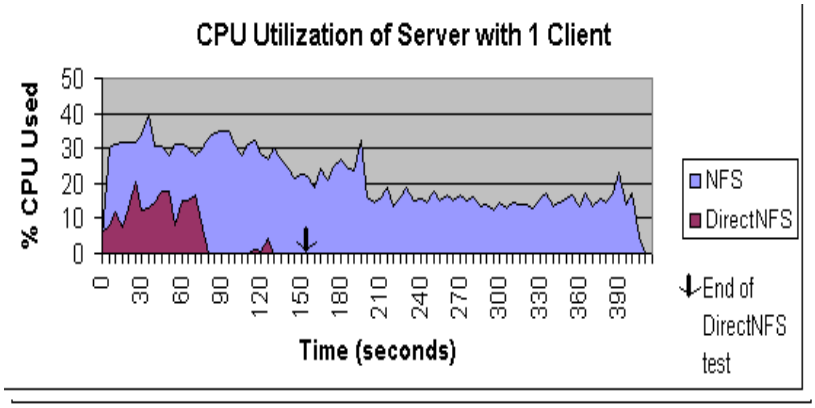


Figure 12: CPU utilization figures for a single client setup

However, a look at the DirectNFS numbers for the same test conditions shows a radically different scenario. One can see that there is an initial period where the CPU utilization is roughly at an average of 10%, with a peak utilization of 20%. This is because of aggressive pre-fetching of metadata by the DirectNFS client during the start of file I/O. This accounts for the lower CPU utilization on the server when servicing a DirectNFS client as compared to a NFS client.

Thus, it can be seen that the CPU utilization is significantly lower than NFS utilization for the same one client setup that we used to measure NFS utilization. This indicates that the DirectNFS metadata server may scale better than NFS servers.

Another key parameter by which scalability can be judged is the amount of network traffic, expressed in terms of the number of RPCs that are required for a given operation to take place. A measurement of the number of RPCs that are required to run the given set of tests reveals that DirectNFS uses about a tenth of the total number that is required for NFS. This can be explained by the fact that the number of metadata requests in DirectNFS is drastically lower than NFS because of write allocation gathering and the metadata pre-fetching performed by the client. This makes the data-metadata split attractive, as this considerably reduces the traffic on the network and makes DirectNFS a lot more scalable.

Overall, DirectNFS performs significantly better than NFS for all of the tests, outperforming it by a factor of 2 to 3.

DirectNFS has been designed to counter network bottlenecks and 'store-and-forward' overheads on NAS servers. So, the server CPU and I/O subsystem are no longer the bottleneck. Introducing parallelism to storage access also means that the system will scale as the available bandwidth for the storage network increases. Isolating storage traffic on to a separate network allows for better utilization of the messaging network by

other network application protocols.

6. Future Work

1. **Client Side Disk Caching:** To further improve performance, the size of the cache that holds the physical block translations should be made as large as possible. To overcome the memory size limitations that we will come across when dealing with large files and clients with multiple such workloads, the block translations can be stored on disk. Thus, the limitation that currently exists on the number of cacheable translations increases greatly, helping us to achieve greater scalability.
2. **Volume metadata caching:** When the metadata server receives a GETBLKLIST request, the DirectNFS filter uses the physical file system's *bmap* operation to obtain the physical block numbers for the requested byte range. Normally, the block buffer cache would cache the most frequently used blocks in the storage system. Servers normally have a large amount of RAM, and we feel that caching the entire metadata for the file volume is feasible. In fact, for a file system formatted with 4KB-sized blocks, the cost of caching all the physical block numbers of the volume is about 1MB per GB.

7. Related Work

There are some interesting existing systems in the distributed File Systems space. Storage Tank [17] follows a similar approach for moving the data access path away from the server. However, the design of Storage Tank lacks the portability of DirectNFS. This is because DirectNFS uses a portable approach leveraging the ability of a code generator like FiST to drastically reduce the porting of the file system to multiple platforms. Many cluster file systems such as the Veritas Cluster File System [18] are layered above and integrated with a proprietary physical file system. CMU's Network Attached Secure Disks requires Intelligent Devices, which embed some file system functionality in the Storage devices thus handling various issues like security, scalability and object management. NASD addresses the security aspects of a SAN based file system well, but the need for manufacturers to incorporate these changes into disks highlights the problem associated with this approach.

Other similar work in the area includes Frangipani/Petal, Tivoli's SANergy [19] and EMC's Celerra[20].

8. Conclusion

DirectNFS presents an optimum blend of NAS and SAN storage technologies. It uses traditional distributed file system protocols such as NFS for meta-data access, with extensions for direct data access using SANs. The end result is a distributed file system that scales much better at high loads and has a data throughput that is a factor of 2 to 3 better than existing NAS protocols. In fact, this performance was comparable to that of a local file system.

The portable design of DirectNFS makes it relatively simple to port to other operating systems. In the future, we plan to port DirectNFS to other platforms such as HP-UX, Windows2000 and FreeBSD and add CIFS compatibility.

Acknowledgments
We would like to take this opportunity to thank Anandamoy Roychowdhary, who played an important role in both the design as well as the implementation of Direct NFS.
We are grateful to Sunu Engineer, who helped with the design.
We would also like to thank Alban Kit Kupar War Lyndem, Tanay Tayal and Gurbir Singh Dhaliwal who helped with the implementation.

References

[1] Sun Microsystems, NFS: Network File System Protocol Specification, *RFC 1094*, 1988.
[2] P. J. Leach, A common Internet file system (CIFS/1.0) protocol, *Technical report, Network Working Group, Internet Engineering Task Force*, December 1997.
[3] C. A. Thekkath, T. Mann, and E. K. Lee., Frangipani: A Scalable Distributed File System., *In Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Oct. 1997.
[4] Kenneth W. Preslan, A 64-bit, Shared Disk File System for Linux, *Proceedings of the Sixteenth IEEE Mass Storage Systems Symposium held jointly with the Seventh NASA Goddard Conference on Mass Storage Systems & Technologies*, 1999
[5] Sun Microsystems., Open Network Computer: RPC Programming., *The official documentation for Sun RPC and XDR.IBM Inc*.
[6] Chet Juszczak, Improving the Write Performance of an NFS Server (1994), *Proceedings of the USENIX Winter 1994 Technical Conference,* 1994
[7] M.G. Baker, J.H. Hartman, M.D. Kupfer, K.W. Shirriff, and J.K. Ousterhout. Measurements of a distributed file system., *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles. pages 198-212*, 1991
[8] D. S. H. Rosenthal., Requirements for a "Stacking" Vnode/VFS Interface", *UNIX International*, 1992
[9] G. Gibson et al., File Serving Scaling with Network-Attached Secure Disks, *Proceedings of the ACM Int. Conf. on Measurements and Modeling of Computer Systems (SIGMETRICs `97), Seattle, WA*, June 15-18, 1997.
[10] Y. Klein and E. Felstaine., Internet draft of iSCSI security protocol. *http://www.eng.tau.ac.il/~klein/ietf/ietf-kleiniscsi -security-00.txt*, July 2000
[11] ANSI, Fiber Channel Transmission Protocol (FC-1), *ANSI draft standard X3T9.3/90-023, REV 1.4,* July 6, 1990.
[12] Erez Zadok, FiST: A System for Stackable File System Code Generation, *PhD thesis. Columbia University*, May 2001.
[13] NameSys Inc., The ResierFS file system, http://www.resierfs.org, 2001
[14] B. Callaghan, B. Pawlowski and P. Staubach, NFS v3 Protocol Specification, *RFC 1813*, June 1995.
[15] ANSI, SCSI-3 Fast-20 Parallel Interface, *X3T10/1047D Working Group, Revision 6*.

[16] W. Norcutt, The IOZone file system benchmark, *Available from http://www.iozone.org/,* April 2000
[17] Storage Tank Software*, http://www.ibm.com/*, 2000
[18] Veritas Inc. Veritas Cluster File System, *http://www.veritas.com*, 2001
[19] Mercury Computer Systems Inc., High Speed Data Sharing among Multiple Computer Platforms, *http://www.sanergy.com,* 2001
[20] EMC Corporation, Celerra, *http://www.emc.com*, 2001

Building a Single Distributed File System from Many NFS Servers
-or-
The Poor-Man's Cluster Server

Dan Muntz

Hewlett-Packard Labs
1501 Page Mill Rd, Palo Alto CA 94304, USA
dmuntz@hpl.hp.com
Tel: +1-650-857-3561

Abstract

In this paper, we describe an architecture, *NFS^2*, for uniting several NFS servers under a single namespace. This architecture has some interesting properties. First, the physical file systems that make up an NFS^2 instance, i.e., the file systems on the individual NFS servers, may be heterogeneous. This, combined with the way the NFS^2 namespace is constructed, allows files of different types (text, video, etc.) to be served from file servers (potentially) optimized for each type. Second, NFS^2 storage is strictly partitioned—each NFS server is solely responsible for allocating the resources under its control. This eliminates resource contention and distributed lock management, commonly found in cluster file systems. Third, because the system may be constructed with standard NFS servers, it can benefit from existing high-availability solutions for individual nodes, and performance improves as NFS servers improve. Last, but not least, the system is extremely easy to manage—new resources may be added to a configuration by simply switching on a new server, which is then seamlessly integrated into the cluster. An extended version of this architecture is the basis for a completed prototype in Linux [5].

1 Introduction

NFS [1] servers are widely used to provide file service on the Internet. However, adding new servers to an existing namespace is management intensive, and in some ways inflexible. When a new server is brought online, all clients requiring access to the new server must be updated to mount any new file systems from the server, and access rights for the new file systems must be configured on the server. Additionally, the new file systems are bound to sub-trees of each client's namespace.

The NFS^2 architecture allows standard NFS servers to be combined into a single, scalable file system. Each NFS server is essentially treated as an object store. New servers added to an NFS^2 system merely add more object storage—they are not bound to a particular location in the namespace. Clients accessing the NFS^2 file system need not be aware when new NFS servers are added or removed from the system. The system takes its name from the fact that NFS is being used "on top of" NFS—the NFS protocol is being used to maintain object stores, and these object stores are combined into a single distributed file system that is exported via the NFS protocol.

2 Architecture

Figure 1 shows one possible configuration for an NFS^2 file system.

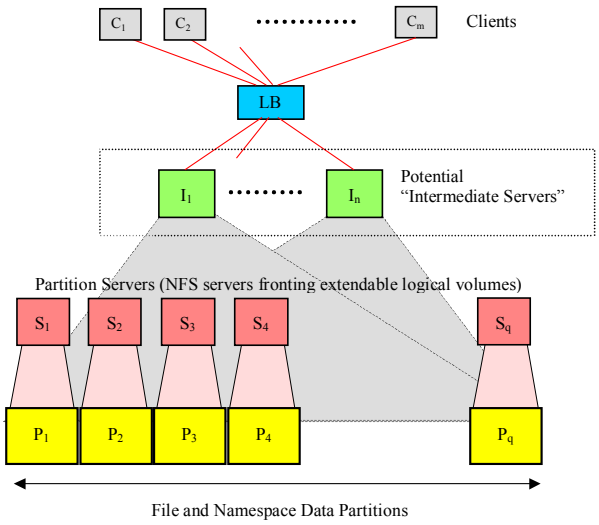


Figure 1: An NFS^2 File System

Storage partitions, P_i, are exported to the other parts of the system via standard NFS servers, S_i, also called *partition servers*. For scalability of the individual partitions/servers, *intermediate servers* can be introduced between the clients and servers. The intermediate servers accept NFS requests from the clients, and transform these requests into one or more NFS requests to the partition servers.

The intermediate servers perform another important, and powerful, function. Each partition server is used as an object store, but some entity must choose which partition is used for the creation of a new object. In the trivial case, this *placement policy* could simply be round-robin. A slightly more complex placement policy could choose a partition server based on resource balancing—choosing the partition server with the most storage available to balance storage resources, or choosing the partition server experiencing the least CPU load to do CPU load balancing. Even more complex placement policies are possible. For example, if one of the partition servers is a slow, legacy machine, and there is some knowledge of data access patterns, less-frequently-accessed data may be placed on the slow machine. This concept could be extended to integrate tertiary storage into the NFS^2 file system.

We describe the architecture under the assumption that the intermediate server translation functionality and placement policy are embedded in the partition servers and that clients issue requests directly to the partition servers. An implementation based on this assumption would retain most of the benefits of the complete system (possibly sacrificing some ability to scale with single-file "hot spots"), but would also have some beneficial simplifications (e.g., reduced leasing overhead, fewer network hops, etc.).

3 Design Considerations

The most important concept behind the construction of the NFS^2 namespace is the *cross-partition reference*. A directory residing in one partition may have children (files or directories) residing on another partition.

There are a couple of alternatives for implementing cross-partition references in NFS^2. Directories in most on-disk file systems are implemented as "files," however these directories have implementation-specific data and interfaces. If we are allowed to modify the NFS servers, directories can be implemented using regular files in the underlying physical file system. While this adds some overhead when compared to using the existing directory structures of the underlying file system, there are also benefits. Directory files may use a variety of data structures (e.g., hash tables, b-trees, etc.), and can surpass the performance of the typical linear list structure used in many systems [5]. More importantly for our purposes, with *directory files* we can extend directory entries to support cross-partition references, independently of the physical file systems. To achieve the goal of using unmodified NFS servers, symbolic links can be used to construct cross-partition references. We first describe the system in terms of directory files (for clarity), and follow this with a description of how the same functionality can be achieved with symbolic links. Fault tolerance and correctness for cross-partition references, without using distributed lock management (DLM), are addressed in another paper [7].

Another alternative is to store directories separately from files. Standard NFS servers are used to store files while separate servers are used for the namespace (either using directory files, or an alternative mechanism). Cross-partition references enable this separation of the namespace from the files. Servers for the namespace could be NFS servers modified to support directory files, a database, or some other construct.

The NFS^2 file system consists of user files and directory files. Both types of files exist as standard files in their respective partitions—a user file, `/usr/dict/words` might be represented as the file `/abc/123` on partition P_3, while the directory `/usr/dict` might be a file `/def/xxx` (containing *directory entries*) on partition P_4. An NFS^2 *directory entry* associates the user's notion of a file or directory name with the system's name for the file/directory, the partition where the file/directory is located, and any other relevant information. For example, some entries in `/def/xxx` could be represented as:

.:/def/xxx:P_4
..:/yyy:P_6
words:/abc/123:P_3

File handles passed to the client contain some representation of the system's name for the object (file or directory) and the partition where the object resides. This information is opaque to the client, but may be interpreted by the load balancer (LB) to direct requests to the correct partition server. Alternatively, the partition servers could be made the sole entities responsible for interpreting file handle information. A request could be sent to an arbitrary partition server that interprets the file handle and may then have to forward the request one "hop" (O(1)) to the server responsible for the object.

In the initial state, a well-known root partition server (say, P_1) contains a file, e.g., "`/root`", which corresponds to the user's view of the root of the NFS^2 distributed file system. The client mounts the file system by obtaining a file handle for the `/root` file as a special case of the lookup RPC.

Let us consider how some operations are handled in this file system. A `mkdir` request from a client will contain a file handle for the parent directory (pfh) and a name for the new directory (dname). A *switch function* is used by LB to direct the request to the partition server (P_x) where the new directory

will reside. The switch function implements an arbitrary policy for where new file system objects are created (e.g., all video files might be placed on a "video server," or the switch function may chose the server with the most available capacity). P_x creates a new file representing dname that has the name dname′ in the physical file system served by P_x. P_x then issues a request to the partition server responsible for the parent directory, P_y (extracted from pfh), to add a directory entry: dname:dname′:P_x to the parent directory file (contained in pfh). If an entry for dname already exists, the operation is aborted and dname is removed from P_x. Otherwise, the new directory entry is added to the parent directory file and the operation completes.

The communication between P_x and P_y could be implemented using the standard NFS and lock manager protocols. P_x first locks the parent directory file, and checks for the existence of dname. If no entry for dname exists, it can issue an NFS write request to add the entry to the parent directory file. The directory file is subsequently unlocked. Alternatively, this communication could take place via a simple supplementary protocol that would allow the locking to be more efficient—a single RPC is sent to P_y, which then uses local file locking for the existence checking and update, and returns the completion status.

File creation is essentially identical to `mkdir`.

The `read` and `write` operations are trivial (referring to the definitions from Figure 1):

write(fh, data, offset, length):

C_i sends the request to LB.
LB looks into fh and directs the request to the appropriate S_j.
S_j issues a local write call to the file specified in fh.

`Read` is similar.

To construct cross-partition references with symbolic links, we can build an NFS^2 cluster as a proof-of-concept as follows. First, each partition is assigned a name (assume P_i, as in Figure 1). The NFS servers then mount all partitions into their local namespace at locations /P1, /P2, etc. using the standard mount protocol. Now, a cross-partition reference is created by making a symbolic link that references the physical file through one of these mount points.

For the example:

words:/abc/123:P_3

An underlying file, `/abc/123`, contains the data for the file, and resides on partition P_3. The namespace entry `words` is a symbolic link in its parent directory with the link contents: `/P3/abc/123`. It is important to note that we are talking about systems on the scale of a cluster file system, so the cross-mounting does not involve a "huge" number of servers. An extension to this work [5] looks at expanding the architecture to a global scale.

4 Future Work

There are several areas requiring further investigation. The performance of the architecture in its various possible incarnations (the symbolic link version, the directory file version, and others) must be studied.

We also want to investigate the potential uses and performance implications of directory files. Directory files were conceived for the NFS^2 architecture to address the problem of providing a single directory structure over diverse underlying file systems, and the need for an easily extensible directory structure. Such benefits may be useful for other file system research. Also, because directory files allow the directory structure to be flexible, they can be used to investigate alternative data structures for directories, alternative naming schemes, new access control mechanisms, and new types of information that might be associated with files.

Due to the structure of cross-partition references, object-level migration should be relatively straight-forward in NFS^2. Migration and replication are two more areas requiring further research.

5 Related work

There has been a significant amount of research and product development in the area of cluster file systems [2,4,8]. Most are based on principles established in the VAXclusters [2] design. These systems use distributed lock management to control access to shared resources, which can restrict their scalability. NFS^2 partitions resources to eliminate DLM [5,7].

Frangipani proposed one of the most scalable DLM solutions in the literature [8]. System resources are partitioned into logical volumes [3] and there is one DLM server dedicated to each volume. This requires using two levels of virtualization: virtual disk and file system. NFS^2 resembles Frangipani in its partitioning of the storage resources for improving contention control. However, NFS^2 uses one level of virtualization allowing decisions for resource utilization and file placement to be made at the file service level. Also, cluster file systems, including Frangipani, depend on their own, proprietary physical file system. NFS^2 is a protocol-level service and can leverage diverse file systems for optimal content placement and delivery. Nevertheless, NFS^2 is complementary to cluster file systems—a partition can be implemented as a cluster file system and can be integrated into a broader file space.

Slice [6] is a system that also uses a partitioning approach, similar to NFS^2. Slice's file placement policies (small versus large files and a deterministic distribution within each class of files) are implemented in μ*proxie*s—modules that forward client operations to the right partition, operating at the IP layer. To make placement decisions, μ*proxies* have to maintain a view of the server membership in the system. In case of reconfiguration, the new membership information is diffused among the (possibly thousands of) μ*proxies* in a lazy fashion. As a result, resource reconfiguration in Slice is coarse-grained; also, file allocation is static for the duration of an object's life. In comparison, NFS^2 can extend the traditional file system namespace metadata to achieve highly flexible and dynamic file placement and resource reconfiguration. However, this requires extensions (even if minor) to the client access protocol. Slice's μproxy idea could be used to transparently intercept client-service communication and redirect it to the appropriate partition server. In that case, μproxies will not need to maintain distribution tables; instead, they will interpret the contents of the (opaque to the client) file handles to retrieve the location of the server for each client request.

6 Conclusions

NFS^2 provides a mechanism for uniting NFS servers under a single namespace. It simplifies management of multiple NFS servers by providing access to all servers through a single namespace (no need for multiple client mount points), and by providing a transparent mechanism for the addition of new servers as the system grows.

This system avoids distributed lock management, which has been a limiting factor in the scalability of cluster file systems. NFS^2 supports heterogeneous physical file systems within the single namespace, whereas other systems have relied on their own proprietary physical file systems. Support for arbitrary *placement policies* to place files on certain servers allows a great deal of flexibility, including placement of files on servers optimized for a given file's content type, load balancing, storage balancing, and others.

7 References

1. Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B., *Design and Implementation of the Sun Network Filesystem*, in *Proc. of the Summer USENIX Technical Conference*, Portland, OR, USA, June 1985.
2. Kronenberg, N., H. Levy, and W. Stecker, *VAXClusters: A closely-coupled distributed system.* ACM Tansactions on Computer Systems, 1986, 4(2): pp. 130-146.
3. Lee, E. and C. Thekkath. *Petal: Distributed Virtual Disks*. In *ASPLOS VII*, MA, USA, 1996.
4. *Veritas Cluster File System (CFS)*, 2000, Veritas Corp., Mountain View, California. http://www.veritas.com.
5. C. Karamanolis, M. Mahalingam, L. Liu, D. Muntz and Z. Zheng. *An Architecture for Scalable and Manageable File Services***.** HP Labs Technical Report No. HPL-2001-173 (7/12/2001).
6. Anderson, D., Chase, J., and Vadhat, A., *Interposed Request Routing for Scalable Network Storage*, in *Proc. of the USENIX OSD*I, San Diego, CA, USA, October, 2000.
7. Zhang, Z. and Karamanolis, C., *Designing a Robust Namespace for Distributed File Services*, in *Proc. of the 20th Symposium on Reliable Distributed System*s. New Orleans, USA. IEEE Computer Society, October 28-31, 2001.
8. Thekkath, C., T. Mann, and E. Lee. *Frangipani: A Scalable Distributed File System*. In *16th ACM Symposium on Operating Systems Principles (SOSP)*, Saint-Malo, France, 1997.

High Performance RAIT

James Hughes, Charles Milligan, Jacques Debiez
Storage Technology Corporation
1 Storage Tek Drive
Louisville CO 80028-2129 USA
james_hughes, charles_milligan, jacques_debiez@storagetek.com
Tel: +1-763-424-1676
FAX: +1-763-424-1776

Abstract

The ability to move 10s of TeraBytes of data in reasonable amounts of time are critical to many mass storage applications. This paper examines the issues of high performance, high reliability tape storage systems, and presents the results of a 2-year ASCI Path Forward program to be able to reliably move 1GB/s to an archive that can last 20 years.

This paper will cover the requirements, approach, hardware, application software, interface descriptions, performance, measured reliability and predicted reliability. This paper will also touch on future directions for this research.

The current research allows systems to sustain 80MB/s of incompressible data per Fibre Channel interface which is striped out to 8 or more drives. A RAIT system looks to the application as if it were a single tape drive from both mount and data transfer. Striping 12 RAIT systems together will provide nearly 1GB/s to tape.

The reliability is provided by a method of adding parity tapes to the data stripes. For example, adding 2 parity tapes to an 8-stripe group will allow any 2 of the 10 tapes to be lost or damaged without loss of information. An interesting result of this research is that the reliability of RAIT with 8 stripes and 2 parities exceeds that of mirrored tapes even though 8 mirrored tapes requires 16 actual tapes and 8 data tapes plus 2 parity tapes only requires 10 actual tapes.

Keywords: RAIT, High Performance, Archive

1 Introduction

This paper describes the RAIT system as designed for the US DoE as a part of the ASCI program. This system is designed to facilitate the long term archive of large quantities of information in the face of potential media failures.

The requirements of the project are three fold,

- Ensure the reliability of large archives

- Compatible with the existing applications
- Transfer the data at a high data rate

The reliability of tape varies from manufacturer to manufacturer. At STK, our high reliability 9840A tape devices have shown to have an average reliability of one permanent read error every 20TB of data read. While this is significantly better than some other vendors, this error probability is not zero, and can never be.

A probability of a read every every 20TB with a 20GB cartridge, means that a cartridge can be read 1000 times between errors. In general, this is not a significantly high number, but when combined with large multi-volume datasets (files that span and/or stripe out to many cartridges), the effects are multiplied.

For example, a 3 TB backup to 9840A with 1.5:1 compression will require 100 cartridges with a native capacity of 20 GB. Simply because of the number of cartridges involved, there is a 1 in 10 chance that there will be a permanent error in writing or reading the data. Since any error destroys the backup or restore operation, the results are catastrophic to the data.

2 Other Methods

For completeness, we mention other RAIT systems and documentation.

First, although somewhat dated, the Storage FAQ [1] discusses the general issues of RAIT and several vendor offerings. For commercial hardware offerings we find Ultera [10]. For software offering we find Computer Associates [9].

In addition, many backup companies offer striping solutions, these include IBM's HPSS, Veritas and Legato [8, 11, 12]. These striping solutions can provide the performance that a RAIT system provides, but does not add additional data protection. When using a striping solution care needs to be takes because striping multiplies the probability of problems. Our system focuses on solving the robustness problem of stripes tape making the result more reliable than a single tape drive.

This paper is focused on the data protection and transparency of a full virtualized RAIT system. We use the term "full virtualized RAIT" to mean a system that completely hides all aspects of the RAIT system from the application. The application only sees a single tape volume with a single volume serial number. The application issues a single ACSLS tape mount and transfers data to a single tape drive [4, 2, 3].

3 Approach

If we compare RAID to RAIT, they are very similar except that tape is a removable media. We have accomplished RAIT by adding parity (as in RAID), but we have extended this to virtualize the removable media and to provide additional redundancy beyond the single parity of RAID.

The approach that Storage Technology took for the ASCI RAIT project is to virtualize the entire tape operation. By "virtualize" we suggest that the application's view of the operation need not be in full agreement with the reality of the operation.

In this system, the application thinks that it is mounting, writing or reading a single volume from a library. In reality, the “virtual volume” does not exist and another group of real cartridges contain the actual information and additional redundancy which contains what the customer actually gets when the data is read.

The reality is that the single virtual tape mount may have resulted in up to 10 or more real tape mounts, and the data that was transfered to the single virtual drive will have parity added and spread to all the real drives.

In RAID-5, there is a single parity drive. Many customers have experienced multiple failures on a single RAID stripe. In RAIT, multiple parities are created so that if there are any multiple failures (up to the number of parity tapes) the data will be intact.

Just like RAID where the application sees no difference between a RAIT controller and a non-RAID controller, the application that is requesting the RAIT tape mount and writing or reading the data has no knowledge of the reality. In general this will allow any application that reads or writes tapes to be able to write RAIT.

3.1 Hardware

All hardware RAIT Systems are designed to interpose a device between the host and the physical tape library and drives. The device presents a virtual image of the virtual tape and the other deals with “reality”. The STK device has 2 basic parts; a mount proxy and a parity generation/checking data path.

3.1.1 RAIT Proxy

The virtual-to-real tape mount operations are accomplished by the RAIT proxy. This is the device that understands the mapping between the virtual volumes and the real volumes. It also manages the creating of virtual tape pools, reconstruction pools, and control of data path.

The RAIT proxy has a database that contains the persistent information necessary to associate the mapping of virtual volumes to the real physical cartridges. This is critical so that applications that use the virtual volumes are hidden from this fact. This database is mirrored to multiple locations and backed up. In addition, to aid in the transportation and introduction of RAIT groups into other RAIT systems, and to act as a final fall-back to ensure that this information can not be lost, this information is also written onto the tapes as meta-data which is hidden from the user.

From the application point of view, it requests a mount from what looks like a standard STK ACSLS mount service. This single volume that the application requested is translated by the RAIT proxy to real volumes, and these real volumes are mounted.

This is a valuable feature in that it allows applications that only know about ”normal” tape volumes to take advantage of RAIT. This completely parallels RAID where the client hosts have no knowledge that the device is RAID.

Once the mounts are complete, the RAIT proxy initializes the data paths with information on where the tapes are mounted, the number of data stripes and the number parity stripes.

3.1.2 Data Path

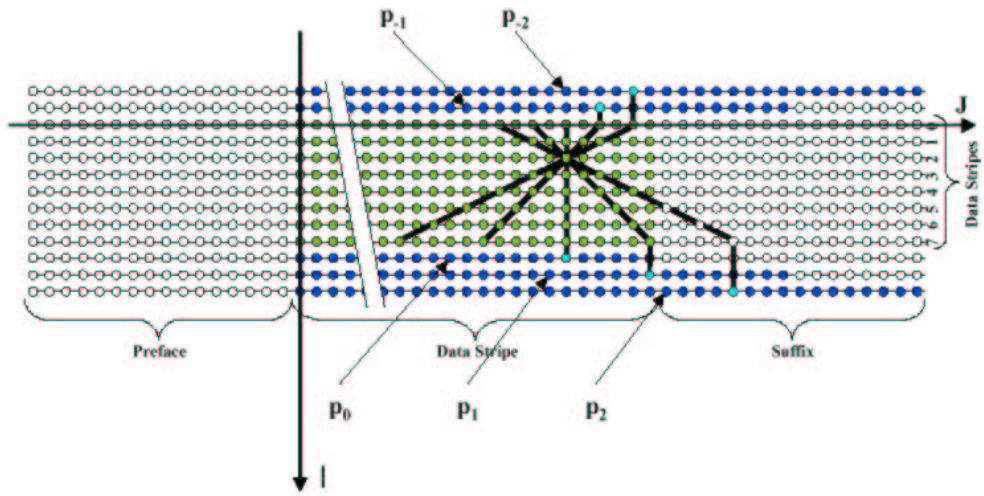


Figure 1: Striping data

The data path stripes the data and creates or checks the multiple parity stripes. Figure 1 graphically shows the representation of how the data is striped and parity calculated.

In this figure, the data is described as a block of **J** words and is striped into **I** horizontal groups. An additional prefix and suffix are added before and after and then the parities are added above and below. In this case there are 5 parities, they are -2, -1, 0, 1 and 2. There are no inherent limit to the number of parities. These parities are described by their row/column slope.

The prefix and suffix contain zeros so that the end-cases of parities which extend beyond the start or end of the user data will have deterministic results. These zeros do not really exist and are not stored on the media, but are included here to illustrate the parity construction.

P_0 calculation is simply the vertical line through the horizontal data stripes. This XOR of the data is stored in the third from the bottom parity stripe. The other parities go through the data and then are stored in their respective stripes.

You will also notice that the parities (other than P_0) are longer than the data. This data is necessary to "bootstrap" the correction function, to keep the blocks independent. This additional data is stored on the tapes. This does not represent a significant lengthening of the parity data. P_x has an additional length of $|x|I$ words. If the blocksize of the tape is 1 MB using 32 bit words on an 8 way stripe, this will result in a 0.003% increase in the length of the parity blocks over the data blocks. Since tape is a variable length block device, this is not a significant factor.

An alternative (and a formal) description; a block of application data that is sent from the host is divided into **I** stripes. Those stripes are sent to the parity hardware to create

multiple linearly independent parity stripes.

The parity generation is accomplished by the creation of vertical and various cross parities. Each of the parities P_k are taken in order from a list $k \; \varepsilon < 0, 1, -1, 2, -2, 3, -3... >$. The data is described as a word $D_{i,j}$ where i is the stripe number and j is the offset.

$$P_{k,j} = \bigoplus_{i=1}^{I} D_{i,j-ik}$$

Where if $D_{x,y}$ is out of bounds, it is assumed to be 0.

3.1.3 Variable Configuration

The configuration of the RAIT volume is selectable. The volumes can be simply defined as N+P where N is the number of data stripes and P is the number of parities.

An optional, and more exact description can be N+(P1,P2) where N is the number of data stripes and P1 and P2 describe the number of parities when failures during write operations are allowed. In this case, P1 is the total number of parities desired and P2 is the minimum number of parities that must be present for the write to be successful.

A simple example: 6+(2,1) will mean that a write of a volume starts out with 8 devices and that during the creation, one can fail and the overall write will be signaled to the host as being good. P2=1 specifies that if there is not at least 1 parities written at all times, then the host will be notified that the write operation has failed.

3.1.4 End of Tape Operations

As data is written, the first real tape device that reaches end-of-tape signals an end-of-tape to the application that this virtual volume is now full. Many years ago, hosts needed to know how many bytes can be written on a tape device. Modern host software assumes that data sensitive compression is occurring on the tape device and no longer needs to know how long a tape is anymore. Programs today simply write data until the tape says “enough”.

Before the data is written out to the drives, it needs to be rotated across the drives because the parities are not as compressible as the user data. Parities are less compressible because when two compressible pieces of data are XORed together, the result is less compressible.

Care must also be taken to ensure that the parity stripes (less compressible) are not written to separate tapes from the (more compressible) user data. Failure to do so will result in the parity tapes always getting to end-of-tape first thus wasting the compression on the data tapes. This is solved by rotating the data and parity stripes over the complete N+P group.

3.1.5 Reconstruction

To eliminate as many errors as possible, we leave all the drive’s error recovery on at all times. This means that all the data integrity features of the device are left enabled. On 9840 this means that Read After Write and full ECC are enabled.

If data can not be read or written correctly the drive notifies the RAIT controller. All retries are used to try to make sure that this is not a transient error.

One important side effect is that, if the data shows up and the drive says there was no error, it can be assumed to be correct. Conversely, if there is an error on read, we can just assume that the data will never be readable and treat that block as missing. Since we know which block is missing, then any one of the parities can be used to correct that stripe.

For instance, if a data stripe is missing and P0 is available, the simple parity of the valid data stripes and the parity *is* the missing data stripe.

Multiple parities are more complicated. For example, if there are 3 missing data stripes and three parities (P0, P1 and P-1) we perform the recovery as follows. Starting with the "correction line" as the first word of each stripe we notice that the top missing stripe can be corrected with the parity stripe going from left bottom to right top. This is because all the words to the left are good (because the prefix is known to be zero) and all words above the top missing stripe are (by definition) not missing. When we are at this case, we can correct the first word of the top missing stripe. We then correct the bottom missing word in the same manner with the other diagonal parity. At this time, there is one remaining missing word and one remaining parity (P0). We can simply use P0 to correct the remaining word. We can them move the correction line to the right by one word.

Subsequent words within this block can be corrected the same way because as we iterate this from left to right, all words to the left of the correction line have been corrected. This simple scheme can be enhanced to any number of errors as long as there are enough parities. When there are more than 3 errors, then the correction line is no longer straight.

These errors correction techniques are discussed in [6] as a "burst erasure channel". A burst erasure is defined as an event where, if there is an error detected, an entire burst (block in our case) is erased (in our case simply not returned from the device). To recover from an error, we simply use the parity to recover the known bad data stripe. IBM introduced the concept of "Crossed Parity"[7], and patents for further extensions to this have been proposed by the authors.

3.1.6 Reconstruction performance

Since this is a burst erasure channel, if all bursts (stripes) arrive without the drive saying there is an error, then we can assume there are no errors and simply reconstruct the user data block without employing the parity hardware at all.

In the case where a single stripe is missing, the parity of everything but the missing stripe *is* the missing stripe. We can employ the parity hardware to create the syndrome in the same time as we took to create the parity in the first place. This allows us to correct a single missing stripe with no performance penalty.

When multiple errors occur, we can use the parity hardware to create partial syndrome for each word and then do the word by word iteration in software.

3.1.7 Additional Data Integrity checks

Provided that all the parity is not needed to correct missing data stripes, the controller can do additional data integrity checks of the data.

3.2 Application software

In general, the application software does not need to understand the operation of the virtual tape devices. The initial customer uses HPSS and the testing of HPSS is accomplished without change to HPSS. Other software such as Veritas or Legato Backup software operates the same whether the tape device is RAIT (virtual) or real.

The one exception is in the area of job scheduling. If the job scheduling system manages the tape drive allocation to ensure that there are adequate resources, this needs to take into consideration that certain tape mounts will not require a single drive, but may require multiple drives. This has been added as a feature to HPSS.

3.3 Performance

The performance of the RAIT system is limited by the speed of the data channel, parity hardware and tape devices themselves. At this time, a 100MBytes/s Fibre Channel is used to connect to the host devices. Fibre Channel can be reliably utilized at 80% of capacity or 80MBytes/s. The parity generator hardware operates at more than 100MBytes/s so that it is not a bottleneck. The devices used are STK 9940 tape devices that have a raw speed of 9MB/s. This number is increased by the compression factor. If the user data is compressible 2:1, then the performance of the tape device will be 18MBytes/s.

A 5+2 RAIT system with 9940s operating with 1.8:1 compression can sustain 80MBytes/s as a single virtual tape device being striped out to 7 physical tape drives.

3.4 Reliability

It has been shown that STK 9840s have a read error about every 20TB of data with a cartridge size of 20GB. Since this is 1000 reads of a single cartridge, a very simple model is to assume a probability of error of $e = 10^{-3}$ per tape operation and unrelated failures [5].

If we assume that a tape operation is a mount and unmount of a tape regardless of the amount of data transfer, then this will be a conservative estimate and the actual reliability should be significantly better than this.

The probability of at least one error for any group of tapes (stripes or just long volumes) is the number of volumes (v) times the error ev. A 100-tape volume has an error probability of 10^{-1}.

A single RAIT virtual tape volume (r) with one parity per 6 volumes (6+1) will only fail if 2 tapes fail. A simple model of this is the probability of 1 of 6 failing and then 1 of 5 remaining fail or $r_{6+1} = 6e5e$ or $r_{6+1} = 3 \times 10^{-5}$.

A (6+2) system will only fail if 3 tapes fail. This is the probability of 1 or 7, 1 of 6 and then 1 of 5 remaining fail or $r_{6+1} = 7e6e5e$ or $r_{6+2} = 2.1 \times 10^{-7}$.

3.4.1 Striped RAIT Systems

Since it is still possible for the application to stripe the data, the application can be used to stripe the data over multiple independent RAIT systems. For instance, 4 RAIT groups at 80MB/s will sustain 320MB/s or more than 1TB/hour.

A striped RAIT system S-wide will have an overall reliability of Sr. In the case of 13 (6+2) RAIT units wide the probability of failure is Sr or 2.7×10^{-6}. This shows that the reliability of a 1GigaByte/s striped RAIT has an error probability of less than two in a million probability of data loss due to unrelated failures.

3.4.2 Other Failure Modes

The analysis in this paper focuses on unrelated drive, and media failures. The performance of the system in the face of related failures at the controller level is not considered. Generally, failures at the controller level do not effect the stored data, which can be read or written once the controller is repaired.

3.5 Future directions

Storage Technology Corporation is in the process of creating a Commercial Off The Shelf device for worldwide availability. STK is also creating a ”mirroring” capability so that tapes can be created simultaneously at multiple locations with the same kind of single virtual device image as RAIT. The performance of the system is also expected to increase as customer systems and tape devices become faster.

3.6 Conclusion

This paper has discussed the method of creating RAIT. The primary goal of reliability is accomplished by adding parity information to the virtual volumes. Performance is increased by striping the data. Further performance can be achieved by striping RAIT systems. In the future this capability will be commercially available.

References

[1] R. Van Meter. *comp.arch.storage FAQ*
http://alumni.caltech.edu/˜rdv/comp-arch-storage/FAQ-1.html”

[2] M. Fisher. *Redundant Array of Independent Tape: RAIT*, THIC, October, 2000, Bethesda MD.
http://www.thic.org/Agenda_1000.html

[3] G. Sobol, *SAN Enabled RAIT/RAIL*, Computing in High Energy Physics, CHEP’00 Padova Italy, Feburary, 2000.
http://chep2000.pd.infn.it/abs/abs_c016.htm

[4] J Hughes, C. Milligan, J. Debiez. *High Performance RAIT*, Computing in High Energy Physics, CHEP’01 Beijing, China, September, 2001.
http://www.ihep.ac.cn/˜chep01/paper/4-004.pdf

[5] R. Defouw, C. Milligan, and T. Noland, *The Level of Data Protection in Redundant Tape Arrays*, Storage Technology Internal Correspondence, May , 2000

[6] W. W. Peterson and E. J. Weldon, *Error Correcting Codes*, 1961, John Wiley & Sons Publishers.

[7] A. M. Patel, *Adaptive cross parity code for a high density magnetic tape subsystem*, IBM J. Res. Develop., vol. 29, pp.546–562, 1985.

[8] R. W. Watson and R. A. Coyne, *The parallel I/O architecture of the high-performance storage system (HPSS)*, Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems, IEEE Computer Society Press, September 1995, pp. 27–44.

[9] Computer Associates,
http://www.cai.com/products/san/saniti_strategy.htm

[10] Ultera Corporation,
http://www.ultera.com

[11] Veritas Corporation,
http://www.veritas.com

[12] Legato Corporation,
http://www.legato.com

Conceptual Study of Intelligent Data Archives of the Future

H. K. Ramapriyan, Steve Kempler, Chris Lynnes, Gail McConaughy, Ken McDonald, Richard Kiang
NASA Goddard Space Flight Center
Greenbelt MD 20771

Sherri Calvo, Robert Harberts, Larry Roelofs
Global Science and Technology, Inc.

Donglian Sun
George Mason University

Ramapriyan@gsfc.nasa.gov
Tel: +1-301-614-5356
Fax: +1-301-614-5267

Abstract

A conceptual architecture study is under way to address the problem of getting the most scientific value from the large volumes of Earth and space science data that NASA expects to accumulate in the future. This involves efficient storage and access, but goes beyond that to facilitate intelligent data understanding and utilization through modeling realistic virtual entities with predictive capabilities. The objective of the study is to formulate ideas and concepts and to provide recommendations that lead to prototyping and implementation in the period from 2010 to 2020. The approach consists of the definition of future scenarios and needs for data usage in applications (in consultation with scientific and applications users), projection of advances in technologies, and an abstraction of an intelligent archive architecture. Strategic evolution is considered in various areas such as storage, data, information and knowledge management, data ingest and mining, user interfaces, and advances in intelligent data understanding algorithms.

1. Introduction

NASA's collections of Earth science data have more than quadrupled in volume since the launch of the Terra satellite in December 1999. At the end of September 2001, NASA's Earth science archives contained over 1,000 terabytes of data and are currently growing at the rate of about 2.8 terabytes per day. Other agencies (e.g., NOAA and USGS) also have large and growing archives of Earth science data. The volumes of Earth science data held by NASA, NOAA and USGS are expected to exceed 18 Petabytes by 2010. Significant increases are expected in the data volumes in space science as well. For example, planned synoptic sky surveys in astronomy could produce 10 Petabytes data per year.

In addition to the large data volumes, there are multiple challenges in managing and utilizing them:

- Data acquisition and accumulation rates tend to outpace the ability to access and analyze them.
- The variety of data implies a heterogeneous and distributed set of data providers that serve a diverse, distributed community of users.
- Human-based manipulation of vast quantities of archived data for discovery purposes is intellectually overwhelming and certainly cost prohibitive.
- The types of data access and usage in future years are difficult to anticipate and will vary depending on the particular research or application environment, its supporting data sources, and its heritage system infrastructure.

Increased hardware capabilities partially mitigate the data access problem. However, adding "intelligence" to the data management and utilization process is essential to automating the end-to-end data lifecycle in order to reduce the burden on data producers and archivists and provide the greatest value to the nation for the data collected. Thus, Intelligent Data Archives here are viewed not just as a set of permanent repositories of data, but also as a suite of services that facilitate the use of data and deriving information and knowledge from them. Therefore "intelligence", in various embedded roles, means the computational transformation of bits into information and knowledge (processing sensory data into models), the ability to automatically act appropriately to complex dynamic conditions (operations automation), and ability to facilitate human interactions with digital resources (semantic management).

A conceptual architecture study is under way to address the problem of efficient access to and effective utilization of the large volumes of data that NASA expects to accumulate in the future. The study is sponsored by NASA's Intelligent Systems Program, and specifically the Intelligent Data Understanding technical area within the program. The intention of the study is to develop ideas and concepts and to provide recommendations that lead to prototyping and implementation in future years. As such, it is not constrained by the need for operational implementation in the near future (e.g., two to five years).

The approach to this study consists of the characterization of future scenarios and needs for data usage in applications (in consultation with scientific and applications users), projection of evolutionary/revolutionary advances in technologies, and an abstraction of an intelligent archive architecture. These steps will lead to a strategy toward the formulation and development of conceptual architectures for intelligent archives. The analysis is used to identify what kinds of intelligent processes are both desirable and feasible, and determine where their application might most effectively drive down costs and enable new applications and research, given anticipated advances in technology. Strategic evolution is considered in various areas such as storage, data, information and knowledge management, data ingest and mining, user interfaces, and advances in intelligent data understanding algorithms.

The following section provides a brief discussion of the preliminary abstracted architecture obtained using this approach. Section 3 presents a description of scenarios and user needs. Section 4 covers projections of evolutionary and revolutionary changes in technology. Section 5 provides a set of recommendations in the form of a road map leading towards intelligent archives supporting intelligent data understanding and utilization.

2. Abstracted Architecture

The abstracted architecture represented here is defined without regard to distributed or centralized nature of implementation and is considered purely from the point of view of the functions that need to exist to support the types of usage scenarios analyzed in section 3. It is possible that with a broader set of scenarios, we will need to identify additional functions in a later version of this abstraction. The functions of an intelligent archive are more stable than the architectures and technologies used to implement them. By abstracting elements and processes into functional elements, we can explore application strategies of technologies and system resources for future intelligent archives.
However, it is first useful to provide our definitions for **data**, **information** and **knowledge**, as these entities are key to the abstraction of the architecture. These are not general definitions, but rather somewhat specific to the domain of scientific research.

- **Data**: output from a sensor, with little or no interpretation applied.
- **Information**: a summarization, abstraction or transformation of data into a more readily interpretable form.
- **Knowledge**: a summarization, abstraction or transformation of information that increases our understanding of the physical world.

Future intelligent archive architectures (see Figure 1) manage these entities with such functional elements as:

- Models and Intelligent Algorithms
  - Consist of models of sensors, resources, data, information, knowledge, and application domain entities (e.g., farm)
  - Include models that exist at multiple levels, ranging from detailed sensor models to models of an entire application domain (e.g., global models in the case of Earth science)
  - Support human understanding of the objects and processes that make up a virtual digital entity and allow users to update the knowledge about the domain as new discoveries are made
- Flow and Feedback Loops
  - Control performance of all other functional elements
  - Include mechanisms that construct, organize, store, update, manage, and provide essential operational services
  - Support self-optimizing operations
- Virtual entity
  - Consists of a representation of the data, knowledge, and processes involved in an application domain

- Provides a context for ingesting, organizing, and managing data and information for the real world entity it represents
- Allows interrogation of past, present, and projected future events, as well as "what if" analyses

- Intelligent information and knowledge extraction
  - Facilitates the transformation of data into information and useful knowledge
  - Automates mechanisms that extract meaning from data and therefore leverage the value of all data in the process
  - Supported by models in the knowledge base, which provide a basis for understanding the data
- Intelligent data production, management and archiving
  - Consists of production, persistence, and active management of valued massive data assets
  - Automates efficient data management mechanisms supporting knowledge-building enterprises in the face of an overwhelming "tidal wave" of data
  - Must dynamically manage high volume inputs from a diversity of observational sensors, converting them into quality usable data products
  - Manages storage close to sensors such that data can be processed locally and passed on to the virtual entity as needed
- Intelligent sensors
  - Are responsible for observations and measurements taken from nature and are the raw ingredients for data
  - Operate from various platforms such as satellites, aircraft, balloons, and in situ constructs
  - Have capabilities for performing autonomous functions and also interact with other sensor systems and external functional elements
  - Include storage, management and processing resources that are part of the overall archive
  - Are modeled in the context of the knowledge base and can support collaborative operation by supplying processing and storage resources when they are not needed locally
  - Are expected to become an integral part of an archive as the architecture becomes more distributed. Here the archive would control sensor data collection based on data needs and would use sensor resources to perform its functions

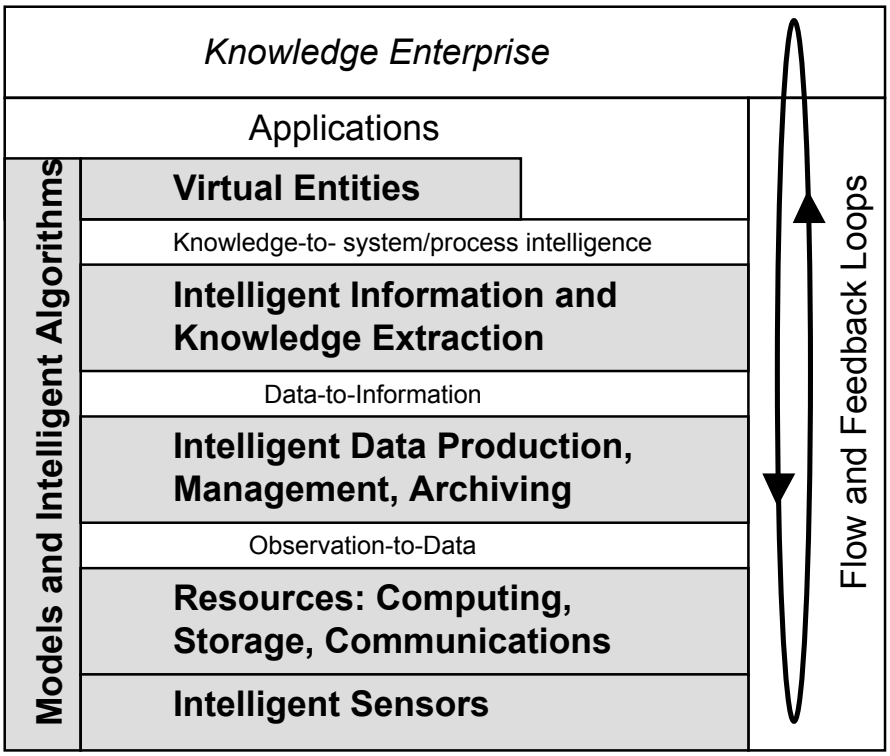


Figure 1: Abstracted Architecture for Intelligent Archives

Using this abstracted architecture to construct an intelligent data system would require a number of design decisions regarding how these elements and entities are represented, such as whether data are represented as bit streams, files, database records, or some other entity. Other decision points concern the relationships among entities, infrastructure to support and connect the various elements, and various optimization schemes. There is much ongoing development in the area of data system intelligence today, such as grid computing, distributed data mining, mobile agents etc. However, because one of the main goals of the abstracted architecture in this study is to aid future research programmatics, the key challenge is to devise an architecture that can be "mapped" into ongoing research and development without being limited to a single architecture evolutionary path.

3. Scenarios and User Needs

We are using a scenario-based approach to the development of futuristic conceptual architectures that enable intelligent data understanding of massive data volumes. Scenario-based approaches are used to drive clear and complete pictures of end-to-end interrelationships among data and information, consumers, data providers, value-added information services, data archives, and data acquisition missions [1,2]. Also, scenario development uncovers a range of requirements for services and capabilities that can be mapped to existing and future technology application. Consequently, forward looking,

tangible and imaginative Intelligent Data Archive (IDA) system application scenarios can be factored into an architectural framework with descriptions of supporting technology.

The scenarios are oriented to an end-user perspective. Scenario descriptions identify "actors" or involved stakeholders and illustrate dependencies among them within an enterprise context. By extension this helps to clarify requirements for corresponding system components and in identifying challenges to be addressed.

Applications scenarios lead to requirements, requirements have implications on technology, and advances in technology affect the evolution of applications. By observing this feedback process, we can characterize several futuristic scenarios. In addition, such a strategy allows the architectural process to adapt quickly to new and evolving scenarios and technologies.

A variety of contexts for possible scenarios have been identified with which to explore, understand, and refine requirements for the IDA architecture. Examples of candidate scenario contexts are:

- Ecological forecasting
- Precision agriculture
- Natural events and hazards (e.g., volcanoes, earthquakes, hurricanes, floods, fires)
- Skilled (10 – 14 day) weather forecasting
- Space weather
- National Virtual Observatory

Of these, in the initial phase of this study, we have used the precision agriculture and precision weather forecasting contexts and developed two scenarios.

3.1 Precision Agriculture

The precision agriculture scenario is concerned with the scope and parameters of a farm employing high-resolution Earth science data. The farm, which constitutes the virtual digital entity in this scenario, is characterized as a relatively small spatial area (considered in acres) for agricultural products suited to regional ecological, weather, and growing constraints.

The “digital farm” concept interrelates ideas about digital technology, digital information, GIS, and human-machine interfaces. We explored potential future requirements and uses of quality high-resolution geo-spatial data employed in precision agriculture. The information resources needed represent the consequence of interoperating services, value-chain processes, automation, and filtering of data of specific relevance to the farmer.

Information-intensive support services helpful for crop planning, cultivation and harvesting include current conditions monitoring, histories and time series studies, trends/risks analysis, prediction, and forecasts, “what-if” investigations, and outcome comparisons. Detailed information about land, weather, water, agriculture markets, prior

yields, agri-chemical options, seeds, etc. are useful for crop planning and planting. High-resolution information is helpful to monitor, assess risks, and make decisions about appropriate interventions to maintain crop health. Similarly, to maximize yields, decisions about harvest timing require information about current and future conditions (e.g., local weather, soil moisture, crop maturity). Remotely sensed information about farm assets, including information collected from the farm about outcomes of plans, cultivation techniques, and harvests, is integrated within a digital farm for long-term use. In all cases it is important that the information be provided to the end-user with confidence estimates.

To make sense of all this information, the digital farm concept includes a digital assistant that works on behalf of the grower and is very intuitive and simple to use. The digital assistant is available from any interface (workstations, mobile devices) from within the house, farm buildings, vehicles, or even the combine. Interaction with the digital assistant can be conducted by natural language either via voice or keyboard.

The digital assistant can interpret, broker, and fulfill requests for information and services from the virtual entity both dynamically and autonomously. In this scenario, the virtual entity is a digital wheat farm that contains encyclopedic farm-relevant information ontologically, spatially, and temporally organized. The digital farm keeps its information stores about soil, crop, weather, and moisture conditions constantly updated. It interfaces with external inputs of data and information sources as well as with farm-specific sensor inputs. These functional interfaces are crucial to pooling farm-relevant data from raw data sources such as primary archives and agricultural services.

The digital assistant can produce different views of this information by summoning an array of functional services. These services can be invoked and combined with an existing farm state model to produce a virtual 4-D representation of the entire farm that the grower can inspect from his or her office or combine cab. The virtual farm serves as an interactive reference of farm-specific assets integrated with historical, current, and modeling information. Views of the farm can be summoned to within a square meter with variable time series. Types of information range from historical to actual current conditions to what-if scenarios cast into the future. Because the grower's digital farm can "learn" from his or her queries and interests, the content and services it provides adapt with change and specificity over time.

Most of the machinery on the farm also interacts with the digital farm information services. Autonomous and semi-autonomous machines that plant, cultivate, and harvest crops are precisely controlled with a combination of GPS, distributed functions, and data from the digital farm. Optimal applications of seeds, fertilizers, and chemicals can be controlled and recorded via wireless digital farm services. Similarly, data taken from the field during cultivation and harvesting can be relayed to the digital farm as input for archiving and further use. Together, the estimated levels of data usage in this scenario approach 650 GB/year for a 1000 acre farm (275GB/year for subset data). Extrapolating the subset data volumes to 600,000 acres of Central Valley agriculture zone in California implies a potential distribution of 165TB/year.

3.2 Skilled (10-14 day) Weather Forecasting

Predicting future weather conditions over a particular region requires accurate data and knowledge about atmospheric forces, physical parameters, boundary conditions, and the interrelated nature of the atmosphere to the physical Earth system. While future knowledge will remain incomplete, scientific processes and visionary methods for improving that knowledge promise more accurate forecasts of atmospheric behaviors as technologies and sensing systems evolve. However, the accuracy of weather predictions tend to decay rapidly as a function of time due to the inability of prediction systems to compensate for noise generated by the chaotic nature of the science, a lack of precise initial conditions and the non-linear complexities of weather.

The weather prediction scenario we considered involves testing a 4-D model of the mid-Atlantic region of North America while studying a developing weather condition. The archival system includes the forecast model and the sensor systems used as input. The strategy used in the forecast scenario is to link the sensor systems with the model such that the archive drives the sensor data collection process. . These sensor systems act in concert, as a web of connected, inter-communicating sensors ("sensor web") [3].

As the system collects data, it creates an initial forecast state that it uses at a future time to compare against actual sensor data. The forecast from the model and the sensor data are compared, and model errors identified. The forecast model is then corrected and a new future state created. This cycle occurs periodically based on forecasting requirements. Employing this closely coupled sensor model process allows short term and long range forecasting with minimal error.

From the scientists' perspective, planning sessions are conducted with an interactive visualization interface equipped with collaborative and immersive human-machine technology. Team members have the option to meet virtually via their workstations or in one of the research center's hypermedia tele-immersive conference rooms. In the tele-immersive room, the scientists plan their research forecasts by summoning a vivid holographic 3-D projection of the Earth, pointing to the region of interest, zooming in, and accessing projections of scaled real-time weather conditions.

The scientists cycle through several current satellite views of the region selected from a list and scan each view. Next they request views of the latest graphics and values for temperature, pressure, humidity, and winds superimposed over the satellite image slightly above the defined region on the global reference projection. In order to assess the whole virtual picture of the weather condition, the team requests that the system detach the selected region from the reference globe and project it as a cube presenting a 3-D visualization of the weather conditions to an altitude of 25,000 meters. By rotating the cube the researchers inspect the sensor grid sensitivities over the region from every angle.

The team next decides to run one hour, one day, five day, and ten day forecasts of weather for this region using the current operational model, adding some custom-selected

inputs from a sensor array pick-list. After a minute, the results are ready to be displayed in the same virtual region cube space. The team studies each forecast display by a variety of interactive real-time commands (by voice, gesture, and keyboard). They explore the 4-D visualizations by varying the temporal resolution, zooming spatial areas/volumes to inspect details, requesting displays of simultaneous analysis result visualizations, and selecting predicted parameters for further comparative analysis. Some team members perform dynamic what-if prediction scenarios comparing what the system generates with their own hypotheses.

With this experience the team then formulates a test of their beta version model using insights gained from the immersive collaborative session. Several on the team notice that higher resolution remote sensing values are needed in certain areas of the region to accurately predict future changes of the pending weather condition. This might accord with the deviation of the standard model from theoretical expectations after one day. Furthermore, there is team consensus that coupling their beta model with selected components of the standard model would elucidate new dependencies and parameters crucial to accurate predictions. Scientist-provided specifications for this new research configuration are then interpreted, translated, brokered, and automatically tasked by the system.

In the final episode of this scenario the team studies the emerging weather phenomenon through virtual projections of real-time information and various combinations of modeled predictions. For the modeling portion of the research, the team observes how the standard model self-adjusts its forecasts as a function of near-real time automated comparison of actual versus predicted parameters. When the predicted varies too much from the actual, new initial conditions are set. This continually keeps the predictive accuracy on track for the near term, but progressive adjustments of the model are required. The standard model in this scenario has intelligence applied so it monitors its own performance. With access to a knowledge base, the model may also pinpoint components to be modified either automatically or by human intervention.

In parallel with this modeling activity, the team custom-configures its beta version model. The team includes a system request that re-tasks the sensor web to gather highly detailed inputs for a critical area of the study region, to generate new forecasts. The sensor web schedules and promptly complies with the request, providing critical detailed data for the beta model to process.

Ten days after the start of the research event, the team is able to conclude from their findings that new knowledge was gained about the rare weather condition. Furthermore, comparisons of performance and outcomes between the beta and standard models identify strong points in the beta model responsible for improving the accuracy of overall forecasts. Validation of these findings leads to the promotion of specific beta version components and two external model linkages to the standard model, adding a new phenomenon to the knowledge base with additional predictive power.

Making the above vision possible obviously involves developing new observation sensor systems as well as innovative techniques for data management and utilization. It is anticipated that improvements to existing capabilities combined with evolving infrastructures and innovative research technologies can enable skilled weather forecasts of ten to fourteen days by 2025 (current forecast predictive skill is five to seven days) [3]. Skilled forecasting goals such as this require quality, mixed-resolution observations and data acquisition systems; very rapid processing of observations; complex data assimilation strategies; predictive modeling strategies and algorithms; and powerful technology infrastructures for archiving, distribution, and interactive visualization. An initial assessment of expected optimized global data volumes covering required parameters, temporal/horizontal/vertical resolutions, and vertical measurement layers yields an estimate of about 20TB/day by 2025.

3.3 Empirical Observations

While futuristic scenarios project the needs for research and applications, empirical observations of data access and usage patterns provide a base state and historical trends. They also give hints on how these patterns may change in the future. The access patterns are a function of the requirements of various users and applications as well as the state of technology. The term technology here includes both hardware and software. For example, existence of faster hardware promotes the use of data mining software, which in turn allows different and more useful forms of access from the archives than is currently possible. As visualization tools, network bandwidths, and desktop computing capabilities increase, new requirements may emerge in accessing archived data.

In the initial phase of this study, we have studied patterns of users' access at the Goddard Distributed Active Archive Center (DAAC) since a record exists starting from the DAAC's inception in 1994. More observations at other DAACs and other types of data centers would be useful to provide a broader insight to access patterns. Some of the questions to be addressed by such empirical observations are:

- Should data products be processed routinely and stored for future distribution, or should they be produced only when a user or an algorithm requests them?
- For data-intensive algorithms, should the data be moved to the software, or the software to the data?
- Should architectures be developed primarily based on average data access requirements or peak requirements, and how can peak requirements be characterized?

A key capability implicit in the term Intelligent Data Archive is an awareness that extends beyond the data. While we commonly think of this awareness in its "operational intelligence" context (e.g., resource management, autonomous data gathering), an intelligent archive should also have "scientific intelligence," i.e., the higher-level knowledge that is derived from the data. Clearly, intelligent archives that include models

have some higher-level knowledge about the data. Beyond that, a wealth of knowledge is published in scientific journals. Studying the connection between data in archives of today and the scientific knowledge derived from them will provide valuable hints for the design of future intelligent archives that embed knowledge with data. This initial phase of study includes a "proof-of-concept" attempt at closing the data-knowledge loop using automated (and semi-automated) methods to link datasets from the Goddard DAAC with scientific knowledge resulting therefrom as expressed in publications (limited to those available electronically). Some of the difficulties encountered here provide valuable lessons in current shortcomings in the world of data archives and electronic publication, which offer opportunities for future work.

4. Technology Evolution/Revolution

In the development of data and information systems over the last ten years, significant progress has been made in several areas. These areas include: handling large volumes of data at high rates, distributed computing, archiving and distribution, data and metadata standards to facilitate system interoperability and provision of services such as subsetting, and user interfaces.

In the Earth science domain, this progress is exemplified by NASA's Earth Observing System Data and Information System (EOSDIS) [4] with its distributed set of DAACs and Science Investigator-led Processing Systems (SIPSs), the NASA-initiated federation of Earth Science Information Partners (ESIPs) [5], and the international Committee on Earth Observing Systems (CEOS).

On a more general level, the Global Grid Forum and NASA's Information Power Grid [6] represent efforts to develop persistent networked environments that integrate geographically distributed supercomputers, large databases, and high-end instruments. These resources are managed by diverse organizations in widespread locations, and shared by researchers from many different institutions. Within the Global Grid Forum, the Jini activity [7] is chartered to address the need for a grid framework to support both resource and service discovery, in an environment in which these resources and service providers may enter and leave the grid dynamically, and where diverse protocols are expected to exist.

It is expected that near-term archiving systems will arise from these efforts as well as several commercial developments in hardware and software technologies. We envision that over the longer term, such "grid" infrastructures will evolve into a finer-mesh, perhaps self-organizing "fabric" as computing and communications become increasingly ubiquitous.

The evolution of (and revolutions in) technology over the last twenty-five years demonstrates the difficulty in predicting the technologies that may be available ten to twenty-five years from today. However, a study of existing forecasts by well-known scholars in various areas relevant to data access and management is useful in conceptualizing new architectures for IDA.

Potential technology drivers include processors, microelectronics, nanotechnology, biotechnology, sensors, intelligent systems, communications, and user interfaces. In each of these areas, advances are being made that will have a dramatic impact on future archive architectures and functionality. In the hardware technology areas, the cost per unit capability has been decreasing rapidly and is expected to continue to do so. The implication of this on the end-to-end data management process and data utilization is that it enables implementation of a number of services that have heretofore been limited by hardware costs and encourages experimentation and advances in software techniques. Advances in techniques resulting from research in intelligent systems (including intelligent data understanding) sponsored by NASA and other organizations become suitable for incorporation into the overall data management and utilization process.

4.1 Advances in Storage Technologies

Today we are witnessing the rapid progress and convergence of the fundamental technologies that make up archiving: storage, computing, and communications. Traditionally, digital storage demands have grown at or beyond 60 percent annually. Over the past several years, growth has exceeded 100 percent per year for Internet and e-commerce applications. Data storage functions have undergone an evolutionary change over the past ten years, and are now commonly performed by smaller high-performance disk drives implementing high-availability RAID storage coupled with more capable archiving software. In addition, magnetic tape technology is continuing to increase in capacity and speed. On the other hand, optical storage now seems more oriented toward the entertainment market. Both storage area networks and network-attached storage (SAN and NAS), along with high-speed optical communication, have fundamentally reshaped the traditional storage model. In addition, SAN and NAS archiving strategies have separated storage from being dedicated to any one server and refocused architectural strategies to implement a union of storage devices interconnected by high-speed optical networks.

Even in the near future (i.e., the next five years), the costs per unit of computing, storage, and bandwidth are expected to continue their rapid decline. The historic trend has been that increases in requirements have kept pace with the reductions in per unit cost to maintain roughly the same annual expenditures for hardware. However, in general, the value of an archive system will move from the hardware to the management and utilization of the data. These are what an intelligent archive should aspire to do as performance and functionality increase, especially in a distributed architecture.

Currently, NASA uses both magnetic disk storage (for on-line access to relatively moderate data volumes) and tape storage (for long-term storage and access to large volumes.) The amount of data available on disk has been increasing as disk storage capacity has increased exponentially over the past ten years (over 60 percent annually since 1992). Indeed, some predict that magnetic disk storage will become more cost effective in coming years even for large volumes, as magnetic tape densities have not been increasing so fast as disk. However, while online storage capacity has increased, our ability to access data has not kept pace because input/output performance has only increased linearly [8]. Magnetic recording for both disk and tape will continue to grow at

about 60% annually until the physical barrier (known as the super-paramagnetic limit) is reached.

4.2 Paradigm Shifts

It is also expected that as a result of scientific advances or fundamental limits of nature, paradigm-shifting revolutionary events are likely over the next twenty-five years. For example, quantum mechanics will play an ever-increasing role because it involves the performance of all microelectronic devices and the creation of molecular and atomic size tools. Today's smallest transistor etchings span a mere 130 nanometers. The expected quantum dimension limit for microelectronics is approximately 25 nanometers, where the laws of quantum physics allow electrons to transition across semiconductor gates even when the gates are closed. In other words, the basis for all modern computing technologies will run into a "brick wall."

The effects of these paradigm shifts are illustrated in Figure 2. In the pre-paradigm shift era, we may have extensions to the architectures of today, with increases in the speed and ability to serve data and information to users. However, in the post-paradigm shift era, the nature of the entire end-to-end system could undergo revolutionary changes. This implies that in conceptualizing IDA architectures, it is useful to think in terms of functional capabilities and their necessary interactions and interfaces without being constrained by today's limitations on the locations of such capabilities.

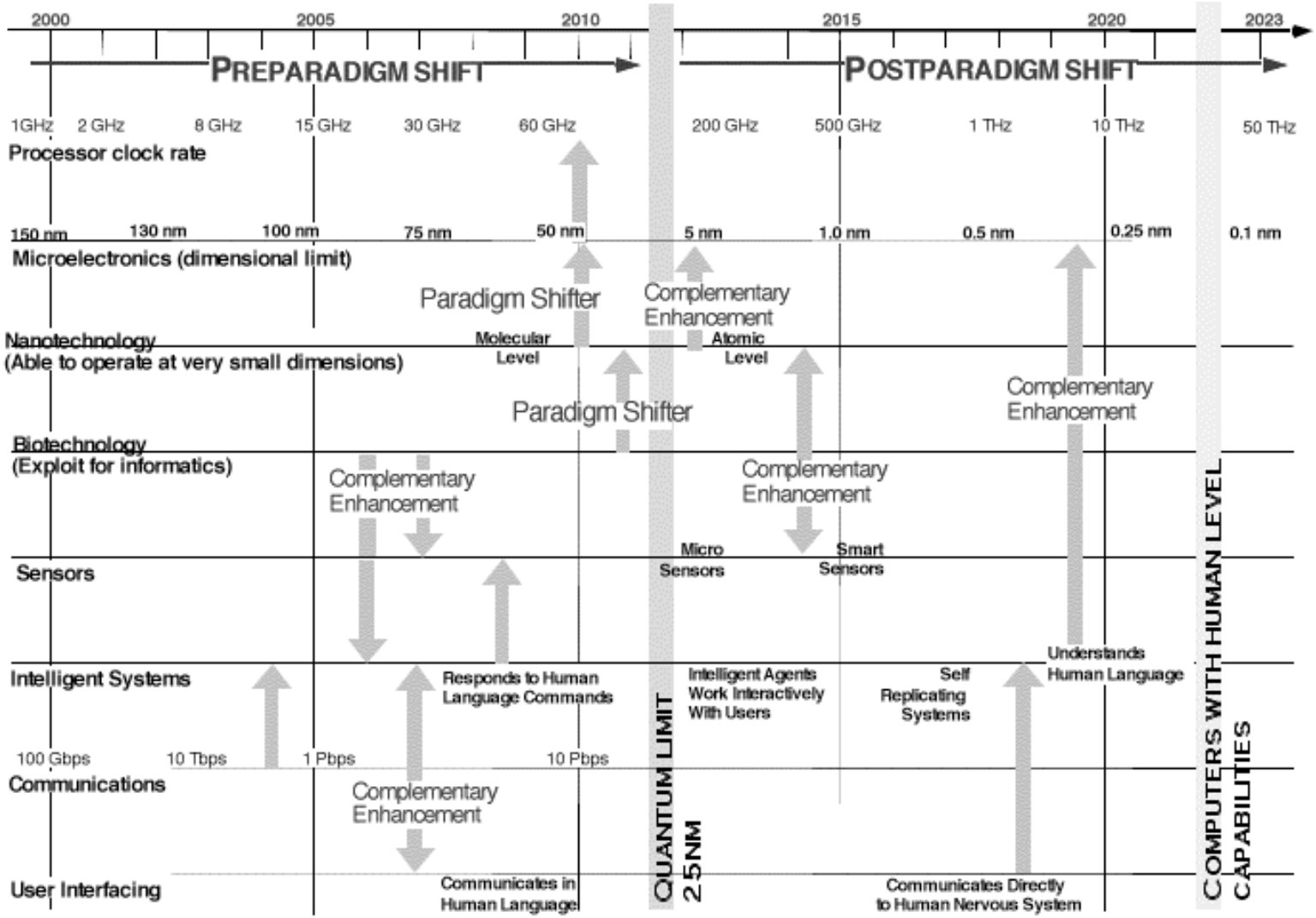


Figure 2: Technology Timeline

5. Recommendations and Future Work

At this stage in our study, we have a set of recommendations shown in the form of a preliminary roadmap to move from archive organizations for traditional data access to intelligent archives that facilitate and take advantage of intelligent data understanding. This roadmap is shown in Figure 3. As shown in this figure, the steps leading to an intelligent archive involve obtaining a better understanding of the following sequence of items:

- Current data access and archiving
- Future data access and archiving trends
- Future scientific applications
- Future enabling technologies
- Roadblocks involved in the formulation, development, and building of an intelligent archive
- Costs involved in formulating, developing, and building an intelligent archive.

There are several areas for further, more detailed, exploration as we continue this study:

- More Scenarios
- It is important to have sufficient scenario diversity to avoid biasing the architecture. Thus, we plan investigation of additional science and applications scenarios in the areas of space science, ecological forecasting, and natural hazards forecasting.

- Specialized Technology
- The initial investigation began with surveying general technologies, such as computing and networking, to determine how they might drive or enable intelligent data understanding. However, there are several areas of more specialized technology, particularly in the area of software, which may be equally important as drivers or enablers. These include areas such as IP-in-Space (enabling a seamless space-ground data system) as well as the various data mining, fusion and visualization technologies being developed as part of NASA's Intelligent Data Understanding program. Advances in science and modeling algorithms are another fertile area.

- Further Architectural Definition
- As the investigation of new scenarios and specialized technologies advances, these should allow further definition and clarification of the IDA architecture. This in turn should promote further definition of the key architectural issues, challenges and trades, which represent an important input into research directions.

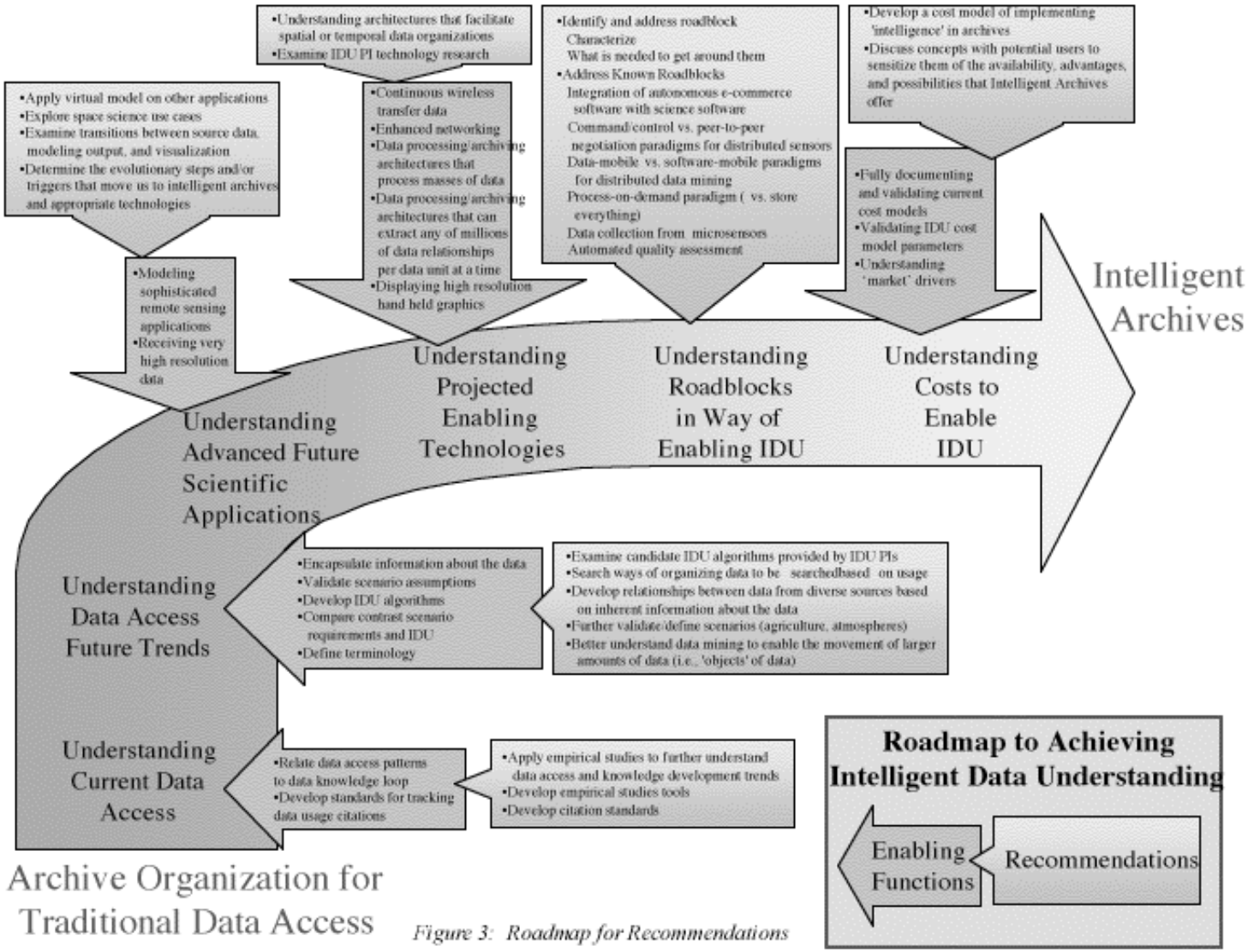


Figure 3: Roadmap for Recommendations

6. Acknowledgements

This study was funded by the Intelligent Data Understanding area of NASA's Intelligent Systems Program. Views and conclusions contained in this paper are the authors' and should not be interpreted as representing the official opinion or policies, either expressed or implied, of NASA or the U. S. Government. The authors would like to thank George Serafino of NASA Goddard Space Flight Center, Kwang-Su Yang of George Mason University, and Randy Barth and Jean Bedet of SSAI for their help with empirical studies, and Lara Clemence of GST for editorial assistance.

References

[1] T. Quatrani and G. Booch. *Visual Modeling with Rational Rose and UML*, (Boston MA: Addison-Wesley, 1998)

[2] Lockheed Martin Advanced Concepts Center and Rational Software Corporation. *Succeeding with the Booch and OMT Methods: A Practical Approach*, (Boston MA: Addison-Wesley, 1996)

[3] M. Steiner, R. Atlas, M. Clausen, M. Kalb, G. McConaughy, R. Muller, M. Seablom, "Earth Science Technology Office (ESTO) Weather Prediction Technology Investment Study," NASA Goddard Space Flight Center, October 5, 2001

[4] G. Asrar and H. Ramapriyan, "Data and Information System for Mission to Planet Earth," *Remote Sensing Reviews*, **13** (1995) 1-25.

[5] The Federation of Earth Science Information Partners, http://www.esipfed.org/

[6] W. E. Johnston, et al. "Information Power Grid," NASA Ames Research Center. Available at http://www.ipg.nasa.gov/aboutipg/presentations/PDF_presentations/IPG.AvSafety.VG.1.1up.pdf

[7] Global Grid Forum. "Charter of the Jini Activity Working Group." March 2001. Available at http://www-unix.mcs.anl.gov/gridforum/jini/charter.pdf

[8] Fred Moore. *Storage IN*fusion. Storage Technology Corporation, 2000.

[9] J. Gray and P. Shenoy. "Rules of Thumb in Data Engineering," Redmond, WA: Microsoft Research Advanced Technology Division, December 1999 (Revised March 2000).

Storage Issues at NCSA: How to get file systems going wide and fast within and out of large scale Linux cluster systems

Michelle L. Butler
National Center for Supercomputing Applications (NCSA)
605 E. Springfield Ave
Champaign IL 61820
mbutler@ncsa.uiuc.edu
Tel: +1-217-244-4806
Fax: +1-217-244-1987

Abstract
This paper will discuss the history of storage at the National Center for Supercomputer Applications (NCSA) over the last fifteen years from inception to a four hundred terabyte archive. The paper discusses supercomputing requirements, hardware and software configurations, and the evolution of data management at NCSA. This paper also discusses the strengths and weaknesses of NCSA's different storage strategies, and gives a detailed discussion of the current system and how it is being evolved to meet the requirements of the TeraGrid computing systems, and large-scale Linux clusters.

1 Introduction

As NCSA, compute power has increased over the years, and so has the mass storage system to keep up with the ever-increasing rate at which data is produced. The NCSA mass storage system started in 1986 with thirty-six gigabytes of disk, a dual processor Amdahl performing twenty MIPS, with fifteen megabytes memory, and a single network adapter in the form of a 1.5 megabits Hyperchannel connection. The system has evolved to a single system configuration of sixteen 250MHz processors, twelve gigabytes of memory, three Hippi and six GigE network interfaces, and two terabytes of disk for overall I/O performance of two hundred megabytes per second.

2 History of Mass Storage at NCSA

In 1986, the first mass storage system at NCSA was an Amdahl running the Common File System (CFS) software package originally developed by LANL. This system was in production from 1986 to 1991 at NCSA, and served an evolving array of supercomputers from NCSA's original Cray XMP, to a Cray2, and a CRAY YMP. Access to mass storage was through a CFS client running on the Cray supercomputers. The data was staged to the Amdahl's disk cache, and then transferred through a proprietary protocol to the compute engine's disk. The only access to the mass storage system was through the Cray CFS client. Disk space on the Cray systems was purged after jobs completed, so users were responsible for storing files they wished to retain. The average file size was skewed by CFS's requirement to break data into chunks of two hundred megabytes. Files could not span tapes, and two hundred megabytes was the maximum that could be placed on the 3480-tape technology employed. All tapes were manually mounted, and redundant copies of every tape were made for off-site disaster recovery. Users began in later years to utilize other smaller data storage facilities. Direct access to their data was needed without mediation by an HSM, and then to a secondary machine like the Crays at NCSA.

The secondary staging was limiting, and the performance through the Hyperchannel was considered extremely slow for the times. User observed data rates were usually 1 Mb/s for a single stream, and multiple streams displayed a more dismal rate. New high-speed tape technologies were emerging, but the Amdahl could not be upgraded to handle those. The Amdahl was neither compatible with emerging tape and network technologies nor capable of advancing to follow on standard data protocols for data transfer.

NSL UniTree and UniTree from DISCOS were researched, and thought to be good products, but support in a 24/7 highly demanding production environment was questionable. Convex ported UniTree to their systems, and created a tuned version that was both faster and met NCSA's reliability requirements. NCSA wrote a conversion program for the move from CFS to Convex UniTree. The CFS databases were converted to UniTree format, and the system was "taught" how to read CFS tapes. Over 2 TB of data were converted, with a downtime of 3 days, to Convex UniTree. NCSA spent the next year rewriting all the CFS data tapes to the UniTree format, so code to read CFS tapes could be deleted at some future date.

2.1 Convex's version of UniTree

In 1991, NCSA moved to a C220I machine from Convex. The machine had dual processors and was wired for fast I/O. It had one hundred gigabytes of local SCSI disk, five hundred megabytes of memory, twelve 3480 tape drives manually mounted, and 1 Ethernet. The main user base still resided on the Cray2 and Cray XMP with a Convex 3880 machine coming into production as an additional compute server. The storage on the supercomputers was still purged as jobs finished, and users were required to store their own files and manage their own mass storage space. Accessibility was changed to a common FTP interface for all data, and data transfer performance improved because of the Ethernet interface(s). At first, the users liked the new procedures and were very happy with the FTP interface but, over time high-speed data networks were installed on the Crays, increasing network bandwidth, and mass storage transfers once again became a bottleneck. The data rate was too slow. User data rates were 6-8 Mbit/s (1MB/s). The one Ethernet interface could not keep pace with 2 systems running Hippi. Jobs were waiting on the Crays, and were wasting compute time in I/O wait states for the mass storage system to return.

The amount of data the system was ingesting was becoming more costly to store, and NCSA was forced to set storage quotas to limit users, mainly by encouraging them to improve their file management rather than by restricting the work they were able to accomplish. However, users reacted by storing their data in alternative, less reliable places that created more hardship for them. A new tape technology, Metrum 2150 tape drive, moved data at twice the speeds of the 3480's, stored seventy times as much on a tape (200 MB on a 3480 vs. 14 GB on Metrum), and a media cost was introduced to alleviate NCSA's storage cost problems. As data was written to tapes holding 14 GB/tape, the media expenditures of NCSA dropped dramatically. The Metrum tape drive specification stated drives should be used over 20% of the day. NCSA calculated that with 8 drives, that requirement could be met. NCSA also still dual-copied all data. The cost effectiveness of the Metrum tape medium enabled NCSA to lift user quotas. Over

the next three years, additional Ethernet interfaces were added with increased disk cache allowing files to reside on disk longer. It became very apparent that a Hippi interface was needed to move data over the network faster, but the C220I machine could not be upgraded to include that interface. The Convex C3880 was being phased out as a compute server, and a large Thinking Machine CM5 was being brought into production. NCSA's mass storage system was "moved" to the C3880 machine. There was no conversion program needed. The C3880 had the same operating system and same hardware as the C220I machine. The databases were moved (FTP) to the new machine along with the tape drives. The data was purged from disk (all written to tape) on the C220I. When the C3880 came up, the data disks were empty, the databases showed all the data on tape, and six terabytes were "moved" to the new machine. All this took place during a normal downtime segment of less than 3 hours.

2.2 Continued Upgrades

The Convex C3880 machine (1994-1997) system was configured with eight Metrum tape drives, two gigabytes of memory, two hundred gigabytes of disk, eight processors, one Hippi interface, and two Ethernet interfaces. All traffic from the supercomputers was routed over the Hippi while traffic from other systems went over the Ethernets. This caused less congestion on the Hippi interfaces for slower data transfers. Users accessed mass storage through FTP and still managed their storage. During the production years of the C3880 archival storage machine, the CM5 was decommissioned, and SGI Power Challenge machines came into production. There was no longer one large machine, but several large machines all running jobs, and storing data. With many more machines capable of storing data through Hippi interfaces, a single Hippi interface could not keep up. Data streams started piling up with 3-4 concurrent transfers, driving down Hippi performance. The Hippi performance from the SGI's to the Convex was poor due to different revisions of hardware. The SGI PowerChallenge machines were capable at the time of 25MB/s, while the C3880 could transfer to the CM5 at 15MB/s, and only 3MB/s to the SGI machines. Tape technologies were also changing. The vendor was phasing out the Metrum tape. Therefore, new tape technologies were needed, but could not be connected on current machine. A new system was needed that could handle multiple Hippi interfaces (the latest revision), numerous simultaneous transfers and, as always, new tape technologies.

2.3 HP Exemplar X-class Machine

In 1997, NCSA purchased for the mass storage system server a HP X-class Exemplar machine. NCSA had stayed on the C3880 machine one year longer because there was not a strong I/O machine to move to until the Exemplar machine was ready for production. There was again very little conversion needed for the twenty-eight terabytes of data to be up and running quickly. The conversion was the same from the C220I to the C3880. All data was purged from disk, databases moved (FTP) showing all data on tape, old host turned off, devices moved, and new host booted with same old name. NCSA stayed on this machine for one and one-half years (1997-1998). This machine had eight processors, four gigabytes memory, five hundred gigabytes of SCSI hardware RAID disk, two Hippi interfaces and three Ethernet interfaces. Our user base started on the SGI Challenge and Power Challenge machines, and then migrated to the SGI Origin class machines. The 2

Hippi interfaces were divided up among the systems so that a "load sharing" could be achieved, giving users dual high speed data transfers into the machine. The new machine was capable of much more throughput than the C3880, so the simultaneous data streams count dropped dramatically. User scratch space was increased and more memory added to the production machines, but data management was handled as previously, an FTP interface for users to move/store data as jobs finished in batch queues.

The mass storage server system turned out to be a terrible environment. HP, who purchased Convex, phased out UniTree and Convex hardware support. Reliability of the system was questionable, it required a reboot every couple of days. NCSA did get some work done in spite of the problems by purchasing six IBM 3590 tape drives including NCSA's first tape robot, an IBM3494 library. NCSA copied all the Metrum data to IBM 3590 tape technology within one year because the vendor was phasing out the Metrum tape technology. The IBM3590 was faster than the Metrum, but did not hold as much data/tape. The IBM 3590 held at the time 10GB/tape. The cost difference was not significant enough to warrant changes in NCSA's storage policies.

The environment for the users remained the same. The aggregate throughput of the machine was much faster, but its instability drew many complaints. The Exemplar machine was able to stage/retrieve user data on both Hippi interfaces at 21MB/s (a combined total of 42MB/s). Normally there were 3 simultaneous transfers, but there have been as many as 12. The number of processors and machines in the Origin cluster continued to climb which in turn increased the need for more data streams to the mass storage system. Stability and aggregate throughput to keep up with the amount of I/O produced by our users were issues and NCSA again needed to upgrade

2.4 The switch to UniTree Inc and SGI

In 1999, NCSA evaluated HPSS, DMF and UniTree, Inc. storage systems. NCSA had a solid base in SGI's technology with much experience in the hardware and the software. UniTree, Inc. was selected to run on an SGI server. A new Origin eight-processor machine was purchased with four gigabytes of memory, two terabytes of locally attached fiber channel disk, three Hippi interfaces, and two Ethernet interfaces. UniTree, Inc provided a conversion program that rewrote the HP formatted databases on to the SGI in UniTree Inc's format, the data was purged from disk, devices moved. The capability to read HP formatted tapes was already in UniTree Inc's version. The new system came up with seventy-five terabytes of data on tape in a matter of hours. UniTree, Inc. on our SGI machine has proven to be reliable and efficient from its deployment in 1999 to today. The aggregate throughput of the mass storage system was 180 MB/s. During that time NCSA's user base was migrated from the one hundred and eighty SGI Power Challenge processors to fifteen hundred SGI Origin 2000 processors logically clustered into 10-15 individual machines. The user data rates were and still are 45MB/s for each stream across the Hippi network. .

The three Hippi interfaces on the mass storage system were load "shared" by dedicating a Hippi interface to the interactive machine, and splitting the traffic for the remaining Origin processors across the other two Hippi interfaces. The six 3590 drives were moved

on to the new system, and a STK Powderhorn with seven 9840 drives and four 3590 drives was installed for a mixed media solution. This is the first time that NCSA has had a "mixed" media tape solution without decommissioning one of the two. NCSA used the 9840 tapes for the smaller files in the archive, taking advantage of the mid-load technology making time to first byte much faster. This small file threshold has changed over the years, but started out as 500MB or less. The 3590-tape technology was used for all other files, and all copy 1 data moved to an offsite facility. NCSA continued to run both IBM and STK libraries until the fall of 2001.

2.5 Upgrades to Origin 2000

Over the last two years, the mass storage system has grown in size and capability. NCSA started with eight 195 MHz processors, two gigabytes of memory, three Hippi network interfaces, and two Ethernet interfaces, an IBM library with capacity for 12 TB of storage, a Powderhorn library with capacity for 120 TB, ten 3590 tape drives, and seven 9840 tape drives. The system today has grown to sixteen 250MHz processors, with twelve gigabytes of memory, an ADIC AML/2 library with two sections for a capacity of 720 TB, an STK Powderhorn with capacity of 120 TB, six IBM LTO tape drives, ten 3590 tape drives, seven 9840 tape drives, eight GigE network interfaces, and three Hippi network interfaces. Its current throughput is 235MB/s with an archive size of 420 TB.

In the past two years, the user base machines have changed. NCSA now has fifteen hundred SGI origin processors with a mixture of 10 TB of disk. There are plans to deploy 15 TB more for production machines early in 2002. The mass storage system today supports a production IA-32 Linux cluster of 1024 processors and five terabytes of disk, a 180 node IA-64 (Itanium) dual processor Linux cluster, and an SGI Origin Array that will be phased out over the next two years as the Linux clusters move into production. The Hippi network will also be phased out, with GigE as the replacement. The performance study that NCSA has completed showed that the 45 MB/s single stream from the SGI's will *not* be matched, but the aggregate throughput of the GigE is greater because the handling of multiple concurrent streams is better. A single Hippi interface single stream runs at 45 MB/s and drops to 25MB/s for two streams, and 8 MB/s for three streams. A single GigE interface from SGI to SGI will transfer data at 25MB/s, and drops to 22 for two streams, and to 20MB/s for three stream. NCSA usually has 5-8 streams of data at all times.

The six TFLOP TeraGrid system will be the next big increment. The data that the mass storage system is ingesting is expected to continue to increase; however, predicting the growth rate and the necessary aggregate throughput needed has been difficult. Big jumps in CPU performance have inevitably produced more and more data, and the growth trends appear to advance along the same curve that is typical of other supercomputer centers. [1] If there is a big jump in CPU hours offered, the amount of data stored shows a proportional jump. But the network bandwidth into and out of the mass storage system that is necessary for applications is hard to predict. NCSA has been increasing aggregate bandwidth of the storage system after the need has been manifested.

NCSA has set a goal for 2002 of achieving 750 MB/s (three times current throughput) as the optimal performance for the mass storage system for the first year of the TeraGrid machine. The Itanium cluster is entering friendly user testing (March 2002). As 180 dual processor machines start storing data to the mass storage system through each systems' own GigE interface, observations will be gathered and adjustments will be made to local disk and archive systems as needed. Only time will tell if these predictions will ring true.

2.6 Hidden work for the mass storage system

The mass storage system at NCSA not only stores/retrieves user data, but also insures the integrity of the data trusted to the archive. In other words, if a file has been stored at NCSA's mass storage system, it will be retrieved. No files transferred properly to the mass storage system at NCSA have ever been "lost" or become irretrievable. There was, on one occasion, Hippi protocol inconsistencies between SGIs that contributed to a handful (<50) of files being corrupted before they reached mass storage. Those files were then retrievable, but still "corrupted". The duplicate copy has been a costly but wise investment. Media failures occur occasionally, but users at NCSA do not notice other than a file might take longer to retrieve than normal. NCSA is constantly rewriting data to new tape formats/media. Migrations in the past have been from the 3480 tapes to Metrum, Metrum to 3590, 3590 to 3590E or LTO, 9840 to 3590 or LTO. When purchasing a machine, NCSA has always included the background processes that need to take place to maintain the environment. Tape drives are not only needed for writing/reading of user data, but for repacking user data onto different tapes, possibly different tape types. The memory, disk cache, CPU, and tape infrastructure must be capable of handling these additional "hidden" tasks of a well-managed HSM.

3.0 Disk strategies for big iron

The large batch systems at NCSA serving supercomputing science over the years have changed quite a bit. Each increase in CPU capacity, memory, and new architectures has meant increased demands on the mass storage system. Sometimes, it has been more bandwidth into the machine for each stream, other times it has just been an increase in the amount of data stored. NCSA has benefited from other disk storage solutions that complement the mass storage system. Pools of local disk for the batch systems, and other smaller disk resources managed by the users for their own data have been highly effective. Each strategy tried has its niche for how it fits in the environment, but none of the solutions can do it all. Below are details on NCSA's file system strategies.

3.1 NFS

NFS has been used by every supercomputer that NCSA has placed in production. The Crays used it for cross mounting file systems to mount home directories and application software. NFS is slow. However it is easy, convenient, stable, compatible, and well understood by users. NFS is currently being used by NCSA for protecting the critical file systems of the large supercomputers. A failsafe server serves file space for user home directories as well as all application software. These file systems are exported from the failsafe system to the Origin Array, the Linux IA32 cluster and Linux IA-64 cluster. NFS is also used to cross mount all the local scratch file systems for each "type"

of cluster. NFS is used by batch jobs to see all storage on the different batch machines, but users take a performance hit by using it for read/write operations.

3.2 Andrew File System

The Andrew File System (AFS) is heavily used more for the desktop infrastructure environment. NCSA hoped in 1994 that AFS would replace NFS for home directories and application software but the file system didn't have the performance required. AFS is used on the Origin cluster for a common link to center-wide installed software such as perl, email readers and the like. Some users do use AFS for data sharing to other environments at NCSA without FTP transfer, but performance is quite limited.

3.3 Local scratch

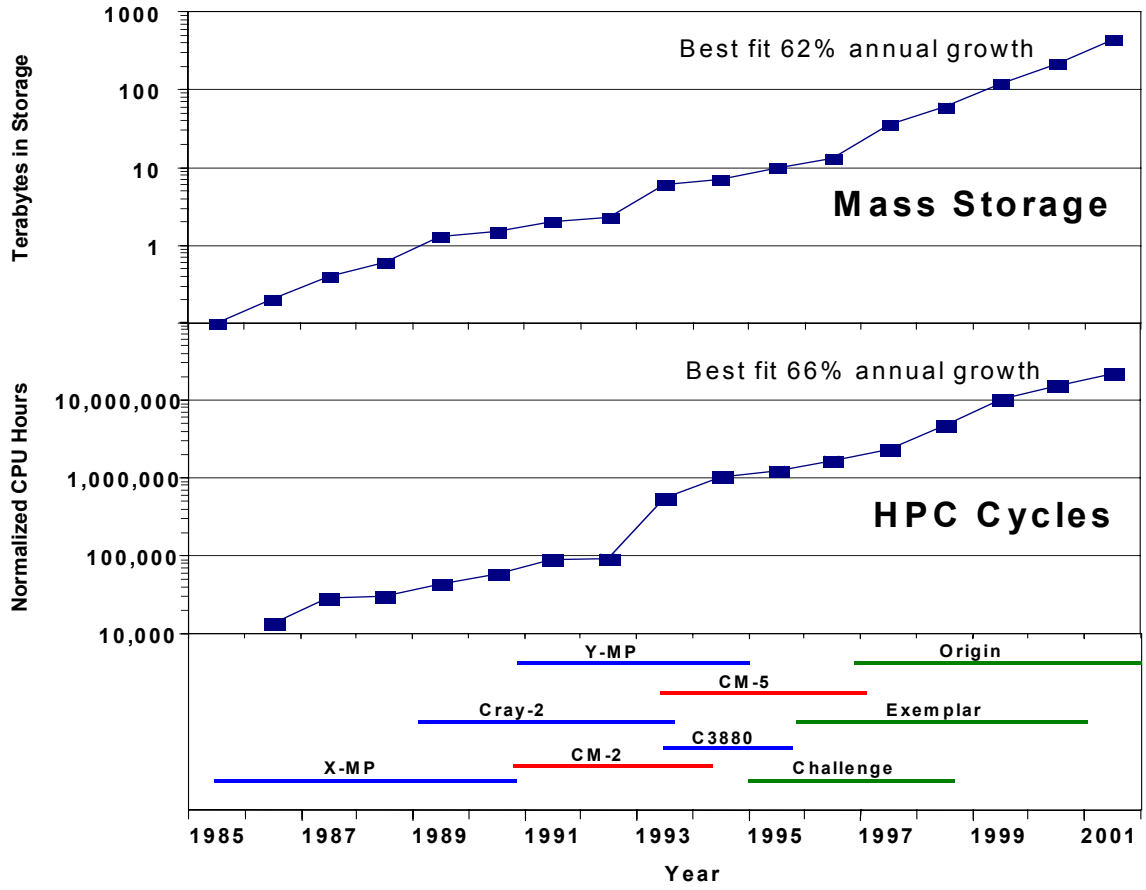
As described above, the large batch systems have local disk attached that is available to users for the duration of their batch job. As the jobs run, data may be retrieved from mass storage and before the job ends users are responsible to store their data back. NCSA has written a few "management" scripts for our users for doing persistent stores so that data will not be removed from scratch file systems until the files actually make it to the mass storage system. In the days of the Cray Super Computers users, had access to a gigabyte of disk storage for scratch space and that has grown steadily to where today NCSA supports file systems in the terabyte range.

3.4 Backup

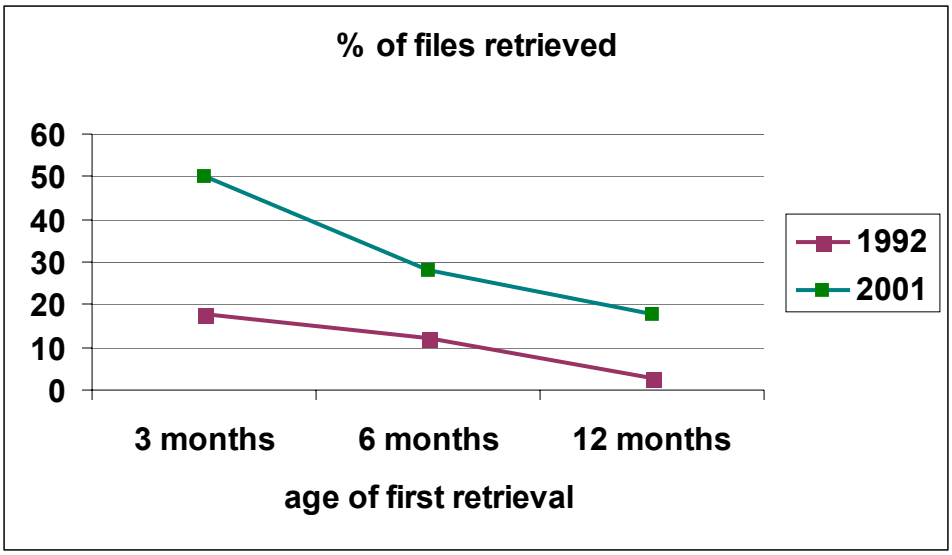
The backup system at NCSA also runs a UniTree storage system on a SUN 6500 machine. It has four IBM LTO tape drives, and shares the ADIC library with the mass storage system. This system handles one terabyte of data per week. NCSA backs up the AFS, NFS, /root, and /usr file systems for all the batch machines and all desktop machine/laptop/file servers. The data in the scratch file systems is too volatile and therefore are never backed up.

4. User and Storage patterns

The amount of storage at NCSA has continued to climb at a steady pace. Recently the growth has been more aggressive. The years 1997 – 2001 saw an 88% growth rate. As machine CPU hours continued to grow at close to exponential rate, the storage also followed faithfully. The chart below maps out the "normalized CPU hours" of the individual production machines at NCSA. The normalized hours have been calculated based on utilization of the machine, and then quantified to be equal among the different machine types. This allows us to equate cpu hours for all machines at the different supercomputer centers for NSF allocations of CPU hours. The bottom section of this chart shows the different machines that were in production during those years.

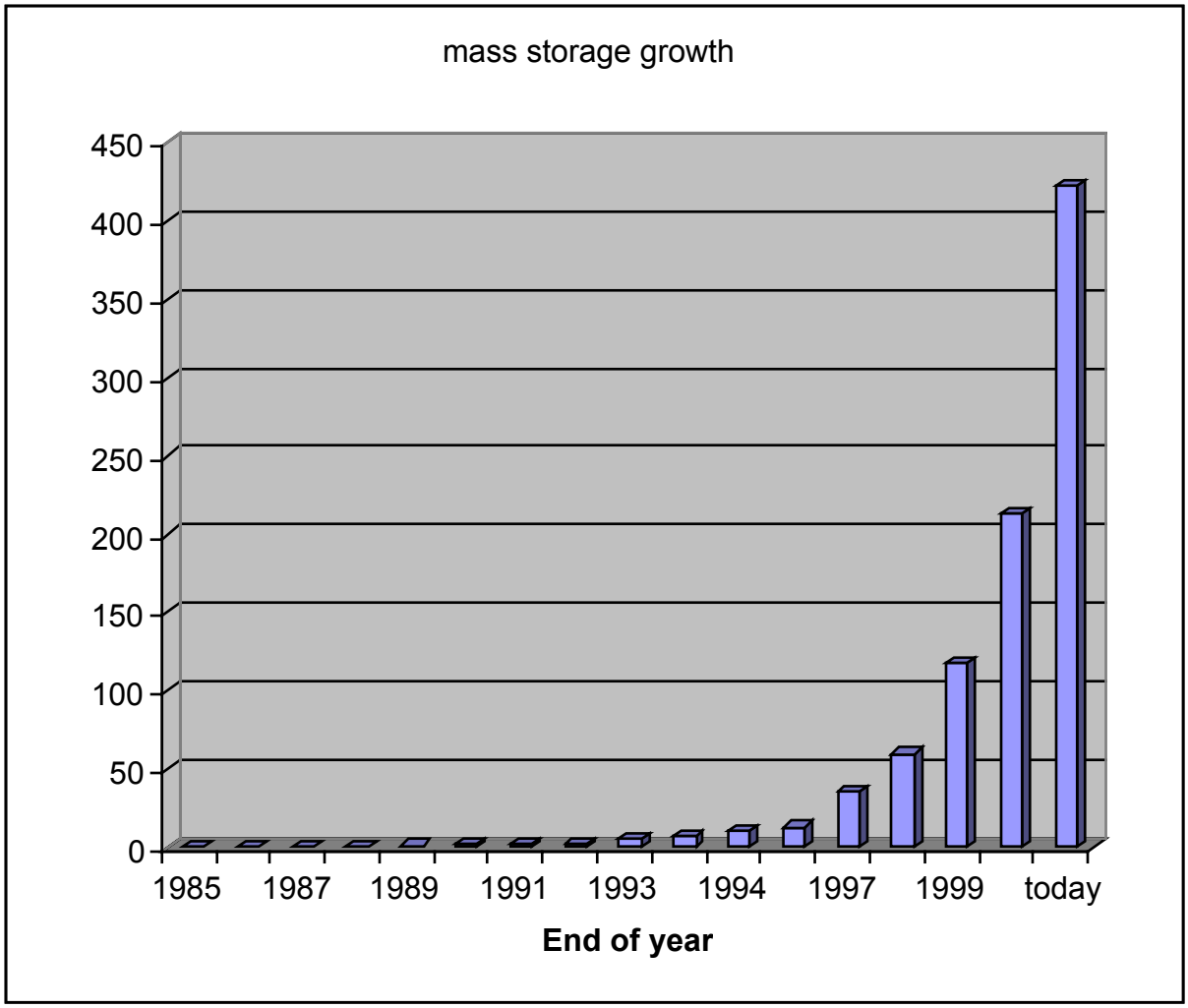


As the archive has grown, storage and retrieval patterns have changed. Large file archives historically have been read only [2] At the CFS conversion time, the size of the archive was 2 TB. UniTree was used primarily to store files that were never retrieved. The older the data, the lower the chance it would ever be recalled. Researchers try to predict what files will be used [3], but over the years, the "reuseability" of the files has changed dramatically. In 1992, as the graph below illustrates, 18% of files up to three months old were retrieved, at six months 12% were retrieved, and after 12 months 3% were recalled. Performance of the archive was unacceptable, and scientists found it faster to recompute data than to get the file from the archive. With increases in bandwidth and stability the data retrieval statistics have been changing, new files in the first three months in the archive have a retrieval hit rate of 50%, the first six months at 28% and drop only to 18% for data within its first year in the archive. So it is no longer a write only archive. Data storage performance was one of the most important criteria that the archive was judged on at NCSA, and now the increased speed and capacity have made data retrieval extremely important as well. Users are no longer recomputing, but retrieving data as needed, quite often, as the chart below shows. As scientific archives grow because of further research data derived from those archives, the role of data retrieval can only increase..

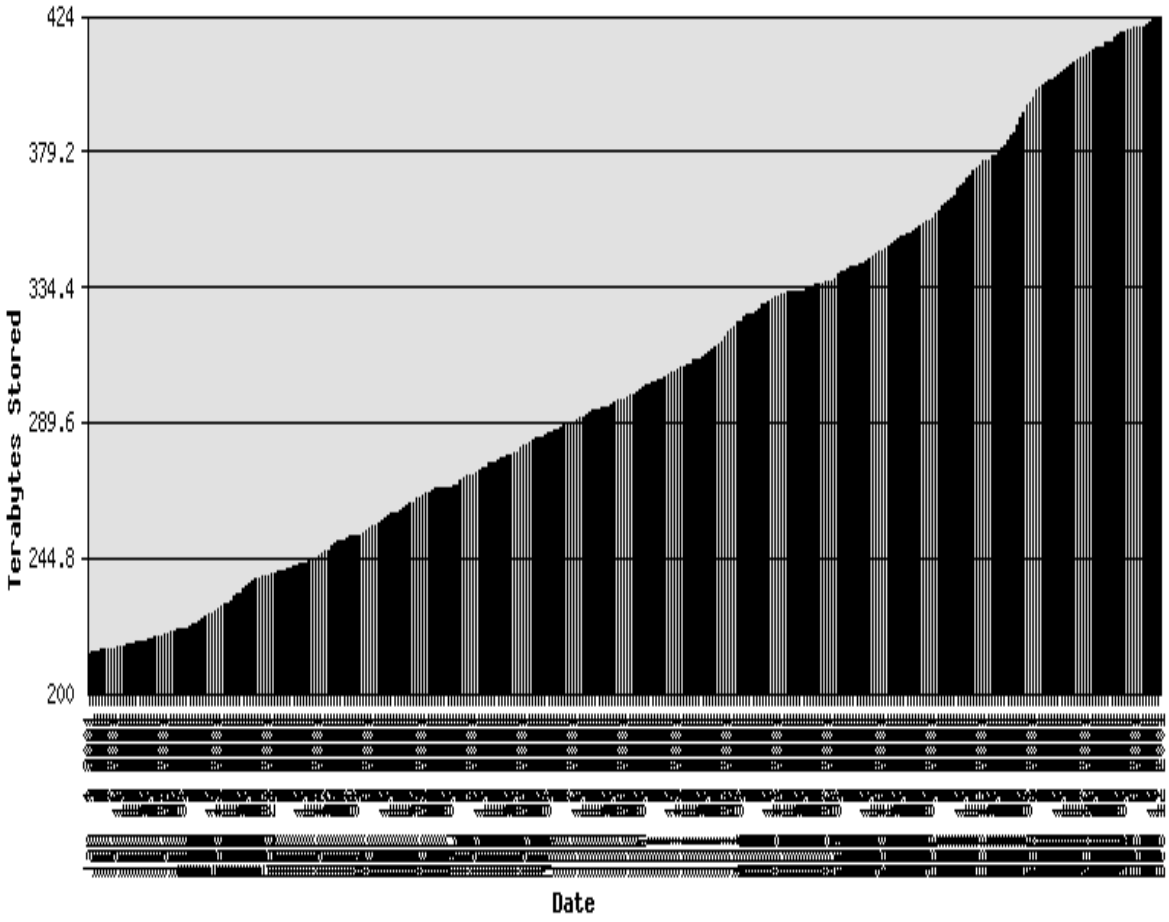


4.1 Growth for whole archive

Our growth patterns have remained much the same over the years. The archive size has been doubling about every year. The NCSA archive by this time next year will be close to a petabyte in size. Below is a graph of NCSA's overall growth. The first ten years are overshadowed on the graph by the huge amounts of data stored in the later years.



The graph of just year 2001 storage statistics for NCSA has a line for each day. The growth is very linear, and continues. For the TeraGrid, there will be a large increase in the data stored, but the amount is not known at this point. It is very hard to predict storage requirements for supercomputer centers [4]. As users have been given more resources in the past, they have produced more data, and storage seems to stay on the same curve as the normalized CPU hours of the machines.. The above graph does show a correlation to the CPU hours of a machine and the amount of data stored, but the number of CPU hours offered by a machine is not known. Within the next five years, there will be a technology switch again, as NCSA continues on the same curve; it is not known what is next for NCSA or supercomputing in general. [1]



4.3 Usage patterns and filesize

The average file size has also doubled in the last couple of years, but the average file size of our archive still seems small for a 400TB archive. Small files are normal for many large archives [4]. A chart of the average file sizes stored in the archive for the last six years shows that it has been increasing, but there are still very small files being used, while there are only a few files that are large (>500GB). This means that when purchasing drives and media types, the small files need to be considered. The small file is sometimes not brought into the mix when discussing mass storage, because large files are the norm, but as seen here, that is not true.

Year	Average File Size (MB)
1996	8.95
1997	13.75
1998	20.49
1999	38.97
2000	43.50
2001	68.88

The filesize growth may be attributed to increased capabilities of the processors so that transfers are no longer as time-consuming. The filesize certainly has not grown as expected, so maybe moving files that are 100GB or larger is still difficult, and a huge undertaking not only to stage, but to work with on the various production machines. As the average file size continues to grow, in 5 years NCSA users will be moving files > 100 GB with ease because of advances in data management and increased bandwidth.

Our top 10 users in FY 2000 stored:

	Files	TB
User1	4,391	3.2 (user 11 in 2001)
User2	259	2.8
User3	77,498	2.5 (user 9 in 2001)
User4	107,722	2.3 (user 1 in 2001)
User5	1,162	1.8
User6	2,743	1.7 (stays in slot 6 for 2001)
User7	3,790	1.6
User8	26,651	1.4
User9	8,757	1.3
User10	9,101	1.2

While in FY 2001 the top 10 users have stored:

	Files	TeraBytes
User1	328,394	9.4
User2	10,163	4.3
User3	23,404	4.0
User4	9,104	3.8
User5	1,871	2.5
User6	4,275	2.9
User7	2,427	2.1
User8	5,683	1.9
User9	30,033	1.9
User10	4,122	1.8

Just among our top ten users, the amount of data stored has considerably jumped. Our largest user in 2000 stored over 3 TB of data in 1 year. In 2001 our top four users each stored over 3 TB of data, with our top user in 2001 alone storing 9 TB. Another interesting point from the data above is that the top users at NCSA do not remain the same year after year. Only 4 users in the top 10 for year 2000 were in the top 11 of 2001.

4.4 Building for the TeraGrid machine

The NCSA mass storage system will be receiving another upgrade in Jan 2002 with an upgrade to six terabytes of disk. NCSA will also add an additional distributed disk server slated for production use in spring 2002. The second disk server will be an SGI Origin 3200 with four processors and two gigabytes of memory. The 3200 machine will have six terabytes of disk and ten GigE interfaces for a throughput of 250MB/s. NCSA is

researching currently how to split data across the machines, with criteria based on uid, gid, original IP address, or file size being investigated. The new system combined with the current system makes the disk cache twelve terabytes with a real aggregate throughput of 450MB/s. NCSA will be also adding ten more IBM LTO tape drives. In late 2002 a 3rd distributed disk machine, an SGI Origin 3400 with aggregate performance of 300MB/s is to be put into production. This will bring the aggregate mass storage throughput to our goal of 750 MB/s. This goal has been based on the TeraGrid machine's predicted performance and the cost analysis of additional bandwidth/throughput for the mass storage system.

Now that NCSA has machines that can handle data at very high rates, and grid and user portal environments are being deployed, improved user tools are needed to move data from place to place. Some important deficiencies relate to inadequate descriptions of what data are available, where the data are located, and how and under which condition users may access the data [5]. The tools that NCSA has given our users have not changed from some form of FTP. NCSA is working on porting GRIDFTP from the Globus group onto the UniTree server so that the FTP transfers will be in parallel to the mass storage system. These tools are also being added to the distributed parallel file systems as explained below. We are incorporating GRID data technologies and working with the Globus group [6] at ANL to enable a grid environment of data being moved, replicated, and archived for all grid users. Gridware from Globus will help users take advantage of different data storage components with in the Grid, and aid the users in data management issues.

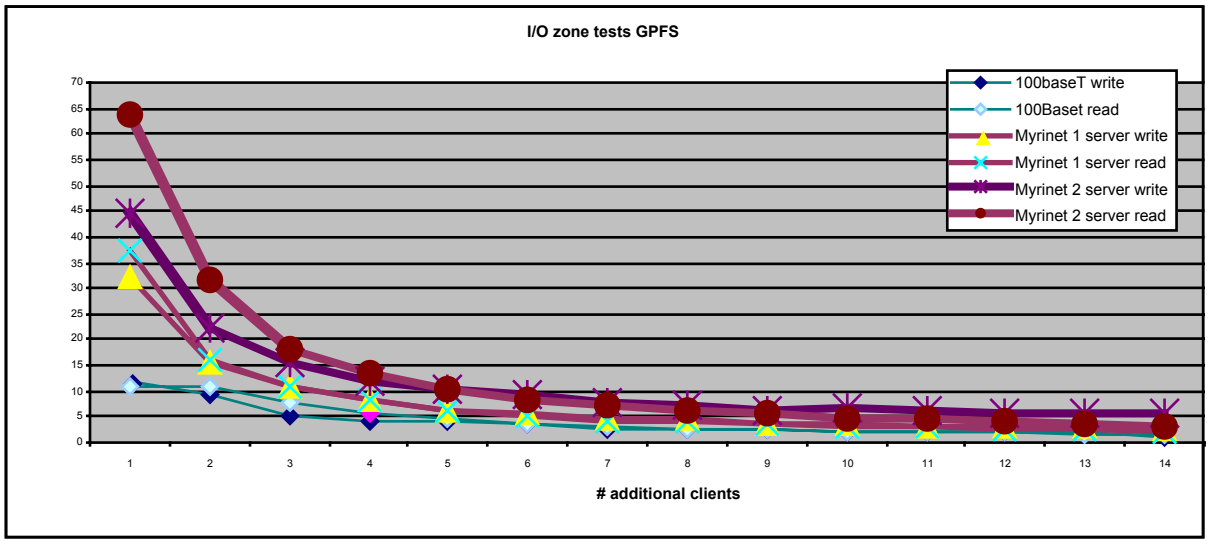
5.0. Linux Clusters Storage

NCSA is looking at many different file systems that might be able to accomplish our goals for the TeraGrid machine, and one standout is the Global Parallel File System (GPFS) from IBM. This is the linux port of GPFS to IA32 architectures from the SP2 machines. GPFS has been running at NCSA since October 2001. GPFS has three major components: a) the disk server is the machine with the disks attached; b) the GPFS server is the metadata server; and c) the individual client. A GPFS file system client must be installed on each system. Each system can then see all the data. GPFS can scale up by adding more servers and clients. GPFS can have multiple servers hosting the *same* file system or individual file systems as needed. NCSA has tested up to 120 clients and 8 servers all seeing the same single file system. GPFS has high availability options so that there is fail-over for disk servers and GPFS servers. Users interface with the native I/O commands to the file system, and all clients can read/write to the same file system and even the same file. Files are distributed across multiple servers by GPFS so that one user can gain access to the entire GPFS file system with all servers writing data at once. The performance does decrease as expected as more I/O requests are added from there.

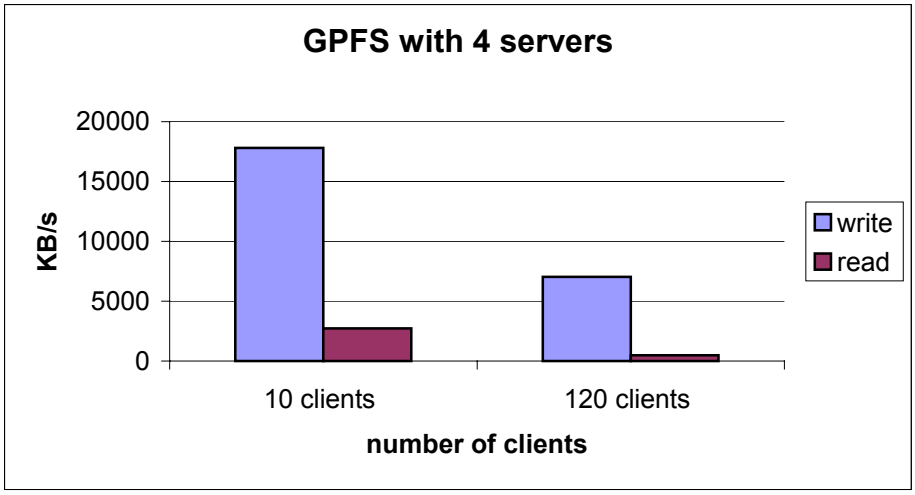
NCSA thinks that this is a very strong product with a very good team behind it. GPFS relies on a very fast low latency network for good performance to be observed. Since the changes in Myrinet driver in release 1.5, GPFS made great strides in reliability. GPFS is a file system for a single system only, there is no data sharing with other systems. A follow-on phase of GPFS development with IBM is a mixed GPFS cluster file system.

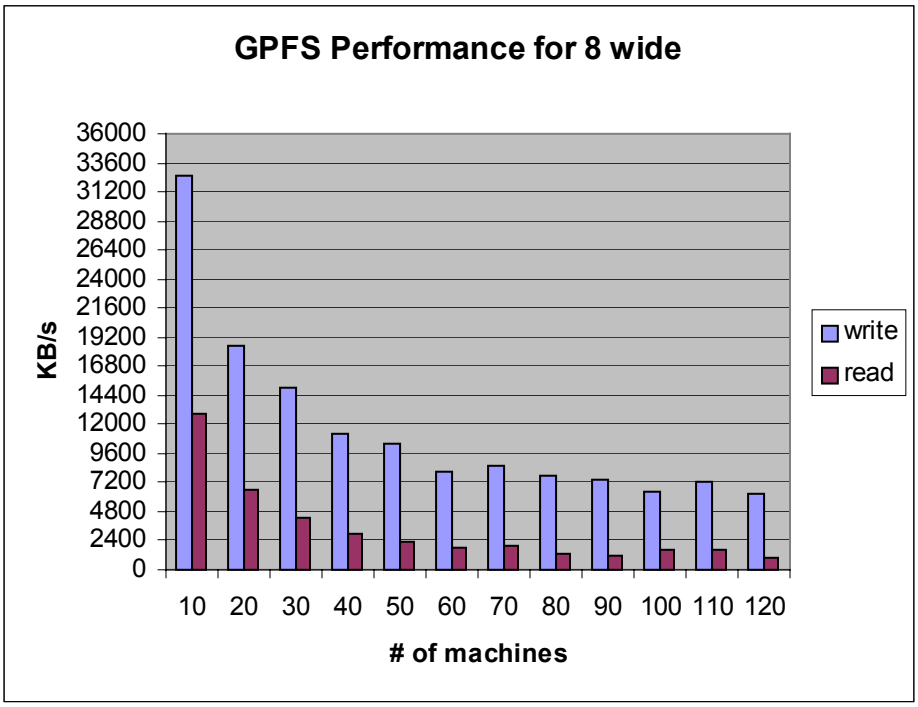
The mixture would be IA64 and IA32 clients and servers for a single GPFS file system. NCSA wants to add the Globus toolkit to GPFS, so that parallel data transfers can be used to move data out of the linux cluster machine to other grid systems or a mass storage.

The chart below compares the performance NCSA had with Ethernet and Myrinet. Myrinet has the best performance. The chart also shows the performance of 2 servers running on Myrinet. The performance that one client receives shows that the single client can gain the entire GPFS file system pipe. The performance scales down from there. These runs on the file system were done before several updated releases of the RedHat kernel with significant I/O changes.



The performance achieved running 4 and 8 servers and various numbers of clients is shown in the next chart. The 4 wide servers numbers were run before tunables for the kernel were made. The 8 wide tests have the kernel mods, but the SAN disks haven't been tuned yet. All clients write a 256 MB file simultaneously**.** Neither IBM nor NCSA is satisfied with the performance, and both are working on that part of this project. Problems are thought to be in the 7.1 kernel. Reads for a 10 wide test of GPFS are >12 MB/s on average, and > 31MB/s for writes.





6.0 Conclusions

The mass storage system at NCSA has evolved over the years. It started out as a small system with a slow interconnect and evolved to a very large system with many fast network interfaces. The supercomputer machines providing the bulk of the data to the mass storage system have also evolved. The machines started out as one system with a few CPU's, changing to a few systems with many CPU's, to many machines with few CPU's. File systems on the supercomputers have also changed, but users must do their own data management. They decide where to put their data depending on their

applications. The interfaces for users to move data are still the rudimentary FTP tools. NCSA is making great strides to incorporate Globus grid tools into clients and servers for utilization of parallel data transfers, and better data management.

NCSA is adding a distributed data cache machine to its mass storage architecture to enable more simultaneous data transfers as the TeraGrid machine is built. More data cache machines will be added depending on how much aggregate data throughput is needed. History has shown that NCSA's data archive is growing at almost the same rate as the normalized CPU hours on the production machines. This is not hard to predict for maybe a year out, but gets harder the farther out one goes. The throughput is the hardest question. Not only do the mass storage archives need to keep up with the production machines on the LAN, but also as GRIDs gain users the amount of data coming in/out from production machines on the WAN will become an issue.

NCSA is looking at many different file systems to provide the best environment for our users. GPFS from IBM is being tested and beginning a friendly user period at NCSA. However more needs to be done to "share" data between these individual compute islands. Moving the data to the machine an application is running on as needed is a step in the right direction, but more needs to be done in this arena. Most of these tools today also deal only in flat files while databases are gaining respect and speed in the supercomputing environments.

References

1. Horst D. Simon, William T.C. Kramer, and Robert F. Lucas, "Building the Teraflops/Petabytes Production Supercomputing Center" *EuroPar* '99 in Toulouse, France, September 1999
2. Heinz Stockinger, Kurt Stockinger, Erich Schikuta, Ian Willers. "Towards a Cost Model for Distributed and Replicated Data Stores". *9th Euromicro Workshop on Parallel and Distributed Processing PDP 2001,* Mantova, Italy, February 2001, IEEE Computer Society Press
3. Timothy Gibson and Ethan Miller, " An Improved Long Term File Usage Prediction Algorithm," *Annual International Conference on Computer Measurement and Performance (CMG '99)*, Reno, NV, December 1999
4. Joshua C Neil, "Characterizing Long Term Usage of a Mass Storage System At a Super Computer Site", *Eighteenth IEEE Symposium on Mass Storage Systems* IEEE 2001
5. CODATA Committee on Data for Science and Technology, Working Group on Archiving Scientific Data, http://www.nrf.ac.za/codata/
6. Ian Foster, Steve Tueke, Carl Kessleman, http://www.globus.org

The Challenges of Magnetic Recording on Tape for Data Storage (The One Terabyte Cartridge and Beyond)

Richard H. Dee
Storage Technology Corporation, One StorageTek Drive, Louisville CO 80028-4274
Tel: +1-303-673-3976, FAX : +1-303-673-8406, richard_dee@storagetek.com

Abstract
Operating points to achieve Terabyte tape cartridge capacities and beyond drive both linear and track densities to values not perceived possible a few short years ago. The primary contributors to the issues related to these high capacities are the physical and magnetic properties of the tape media itself. The total magnetic moment of the recorded bit, driven by the magnetic coating thickness, dominates the recording process and determines the linear recording density possible. Moving a thin tape at high speeds and the mechanical stability in the cross track direction provide engineering challenges for increasing track densities in combination with many parallel channels for high data rates. These issues and trade offs are the main focus of this paper.

1. Introduction

Storing and retrieving data on magnetic tape is driven by (a) capacity (Gbytes/cartridge) primarily because of the cost of storage ($/Gbyte), (b) data rate (Mbytes/second) as people don't want to wait forever and (c) reliability (the data has to be there!). This paper complements the presentations given by Ted Schwarz in past years [1-2] with a little more technical depth. The capacity of a tape cartridge is simply the areal density of the data multiplied by the area of the media used but is often preferably computed in tape by using the relation

$$C = \frac{NbL\varepsilon}{8} \qquad \ldots (1)$$

in bytes, where $\boldsymbol{N}$ is the number of tracks across the tape, $\boldsymbol{b}$ is the linear recording density in bits per inch, $\boldsymbol{L}$ is the length of the tape (in inches) and $\boldsymbol{\varepsilon}$ is a formatting/ECC overhead efficiency factor (typically about 0.7). The 8 assumes 8 bit bytes. The date rate is given by

$$D = \frac{nbV\varepsilon}{8} \qquad \ldots (2)$$

in bytes/second, where $\boldsymbol{n}$ is the number of parallel channels used and $\boldsymbol{V}$ is the speed of the tape (in inches/second). These two relations capture the main parameters in increasing capacities to terabyte levels and data rates to 100's of Mbytes/sec. The linear density ($\boldsymbol{b}$) appears in both calculations and thus is a strong contributor to the problem. The number of tracks ($\boldsymbol{N}$) in the capacity and number of channels ($\boldsymbol{n}$) in the data rate are parameters that may be in conflict when radically increased as will be discussed later.

Capacity (TB)	**0.5**	**0.5**	**1**	**1**	**5**	**5**	**10**	**10**
Data Rate (MB/sec)	**60**	**120**	**110**	**220**	**150**	**300**	**280**	**559**
No. of PII Data Channels, *n*	16	32	16	32	16	32	16	32
No. of Data Tracks, *N*	768	768	1344	1344	4750	4750	4140	4140
Trk. Pitch (μm)	14.0	14.0	8.0	8.0	2.3	2.3	2.6	2.6
Channel Pitch, c_p (μm)	109	55	109	55	109	55	109	55
Rd. Track Width (μm)	7.0	7.0	4.0	4.0	1.1	1.1	1.3	1.3
Tape Speed, ***V*** (m/s)	4.8	4.8	8.0	8.0	9.0	9.0	10.0	10.0
Bit Density (kbpi)	224	224	248	248	298	298	500	500
Track Density (tpi)	1812	1812	3172	3172	11211	11211	9771	9771
Areal Density (Gb/in^2)	0.41	0.41	0.79	0.79	3.35	3.35	4.89	4.89
Bit Cell (nm)	114	114	103	103	85	85	51	51
Bit Cell (ns)	23.7	23.7	12.9	12.9	9.5	9.5	5.1	5.1
Write Eq. Pulse (nS)	9.5	9.5	5.2	5.2	3.8	3.8	2.0	2.0
Tape Length (m)	865	865	865	865	1000	1000	1400	1400
Write Time per Cart. (min)	144	72	152	76	550	275	604	302

Table 1. Terabyte operating points

Table 1 shows scenarios for a 0.5, 1, 5 and 10 Terabyte capacities for various data rates for a normal IBM3480/STK9840/LTO/DLT ½ inch wide tape cartridge form factor. Some tradeoffs between the parameters given in equations 1 and 2 have been included for illustrative purposes and one can easily see where a different set of trade offs could yield the same result depending on which aspect of the tape system you wished to stress more. The stress points are boxed for the cases shown and it is these challenges that are discussed below in relation to the media aspects, the heads and the channel in order to accomplish these operating points.

2. Magnetic Recording

Figure 1 shows a block diagram of a tape recording system from data in from a host computer channel, onto and off the tape and back to the host upon a data read [3]. This figure summarizes the main components and systems needed for the tape system to function. All the subsystems (write method, read equalization and detection, servo, head and tape handling) serve to deal with the unique properties of the tape media itself. This is from both a magnetic and mechanical perspective. The media dictates how the rest of the system is designed in order to achieve high-density data recording and thus is the main contributor to limitations thereto.

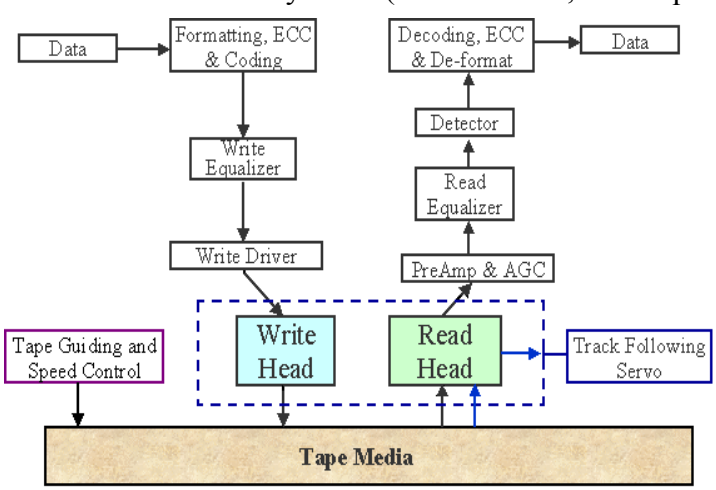


Figure 1. Block diagram of a tape recording system

Fundamental to recording digital data on magnetic tape is the analog magnetic recording that takes place between the

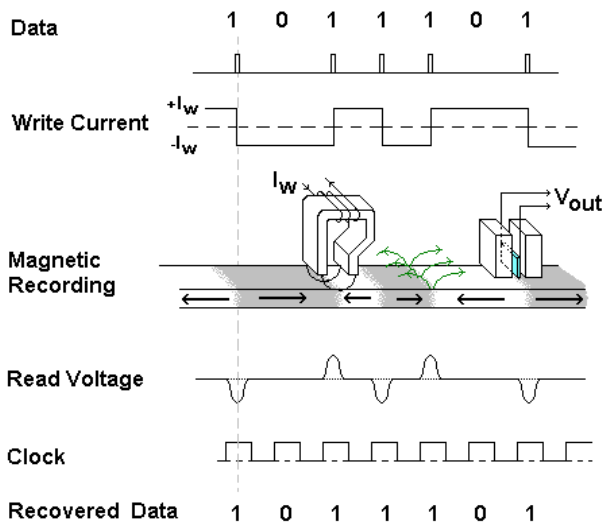


Figure 2. Magnetic Recording

head and the media. These two magnetic components in combination can make or break a reliable data recording system. Figures 2 illustrates magnetic recording on tape and its digital interpretation. The digital interpretation is that a transition between a region on the tape magnetized in one direction to the opposite direction is interpreted as a logical '1' and the absence of the transition a '0' when referenced to a data clock. This interpretation depends on the logic used by the detection system and coding design. For instance a PRML channel (Partial Response Maximum Likelihood) interprets the recorded transitions in a different way by partial amplitude sampling in order to increase the bit density using somewhat lower magnetic transition densities than in straight peak detect channels as illustrated. Such channels increase the logical bit density up to twice that of the recorded magnetic transition density.

3. Recording Technology Challenges

Fundamentally, an increase in linear recording density requires the transitions to be closer and closer together on the media and the ability to resolve them. Table 1 indicates the length of a logical bit (bit cell (nm)) for the various scenarios given for reference (~50 – 100nm). Tape media to date has had the magnetic coating somewhat thick (0.5µm or more) compared to these dimensions which gives broad written transitions due to the generation of transitions curving into the depth of the magnetic coating and the demagnetizing effect of sizeable opposing magnetic poles. These effects are summarized in the equation for the transition length parameter (the 'a' parameter) thus:

$$a = 2\left[\left(\frac{2}{\sqrt{3}}\right)\frac{M_r\delta}{H_c}\left(d+\frac{\delta}{2}\right)\right]^{\frac{1}{2}} \qquad \ldots (3)$$

where $\boldsymbol{M_r}$ is the remanent magnetic moment of the medium, $\boldsymbol{\delta}$ the magnetic thickness, $\boldsymbol{H_c}$ the magnetic coercivity of the medium, and $\boldsymbol{d}$ the head to tape spacing. This relation comes from assuming that the transition follows and arctangent function shape [4]. In order to reduce this transition length parameter the ratio $\boldsymbol{M_r\delta/H_c}$ must be reduced. This can be done either by increasing the coercivity, $\boldsymbol{H_c}$, which physically means it is harder to push the magnetized regions apart or by reducing the medium thickness, $\boldsymbol{\delta}$, which lowers the total magnetic moment and hence the force which is pushing the regions apart.

Reducing M_r is a little more difficult using iron particles (as currently used in MP tape), as this would mean reducing the number of particles in the magnetic coating, which would have the side effect of reducing the signal-to-noise ratio (SNR). An acceptable reduction in M_r could only come from a different particle; for example barium ferrite (BaFe) or a different media construct (such as thin film media). The coercivity of tapes is in fact on the upswing with prototype MP media pushing 2500 Oe compared with today's 1650 Oe 9840 media and 1850 Oe DLT/LTO media. Figure 3 shows how linear density has indeed gated tape products in the past according to media coercivity together with a projection for future systems based on published roadmaps. (The data here are taken from existing IBM, STK, Quantum DLT and LTO tape products). Excessive increases in coercivity would however begin to challenge the available magnetic pole materials used in the write head where the saturation flux density is limited. This would eventually degrade recording performance if the coercivity increases much beyond 3500-4000 Oe.

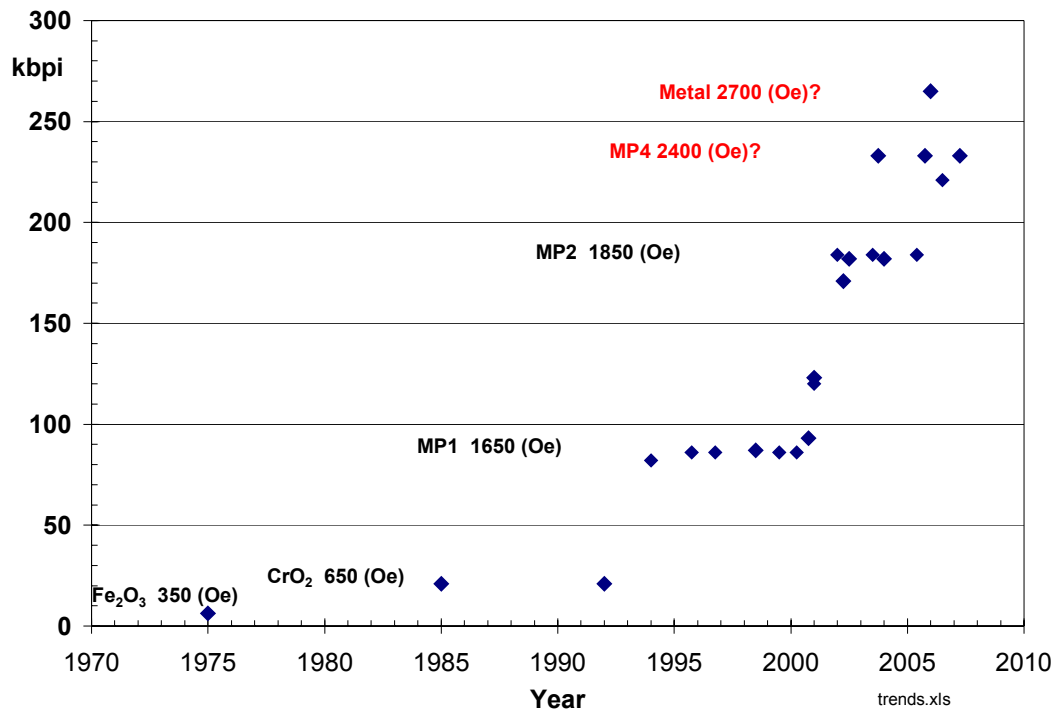


Figure 3. Linear density versus year for linear tape systems

Reducing the thickness is the primary direction to pursue and recently this has been achieved in particulate media by using a dual coating process. Here the magnetic portion of the tape coating is spread thinly over a simultaneously coated non-magnetic under layer. This effectively provides a thick physical coating for smoothing purposes coupled with a reduced thickness magnetic layer as illustrated in figure 4. This has enabled coatings to be produced as low as 100nm and progress is being made to reduce this further [5]. This technique, however, will eventually run out of steam for the particle in binder tape medium concept. One quickly gets to very few 20-30nm thick particles stacked on top of one another in a <100nm coating with the resultant SNR reduction. For areal densities greater than a few Gb/in^2 , the move to thin film media will have to be

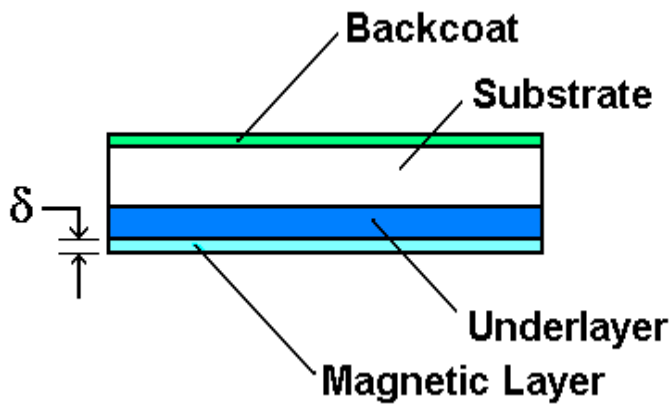


Figure 4. Diagram of a cross section of dual coat tape recording media

made as it was for magnetic disk. (Tape is indeed fortunate that magnetic disk has already demonstrated solutions to high areal density magnetic recording.)

The other parameter that figures into the areal density is track density. Again the number of particles contained within the bit becomes squeezed as the track narrows. As the SNR is related to the total number of particles contained in the bit volume [4] an estimate for the areal density limit, $A_{\mathbf{lim}}$, for metal particle tape can be made from an SNR standpoint and input from media producers on what might be the maximum particle density (smallest thermally stable dispersible particle). Following Mallinson [4] it can be shown that

$$A_{\lim} = t^{\frac{1}{2}} \left(\frac{2pSNR}{3} \right)^{\frac{1}{2}} \qquad \ldots (4)$$

where $\boldsymbol{t}$ is the track density, $\boldsymbol{p}$ the magnetic particle density in the media and ***SNR*** is the signal-to-noise ratio requirement. Using for example 3000 tracks/cm (7620tpi), 10^{17} particles/cm and 20dB we get an areal density of approximately 10Gb/in^2. This assumes that the whole written track is read, no spacing loss and one logical bit per transition. Using a write wide read narrow scenario, as linear tape currently does, and invoking a PRML channel you come out with a very similar number or maybe slightly higher depending on the ***SNR*** and desired raw bit error rate. (PRML channels operate at lower effective SNR values.) The areal densities in the cases shown in Table 1 approach 5Gb/sq.in. and the question arises as to how close to the computed limit can you engineer particulate media for this, or is thin film media prompted as it was in disk.

The other main parameter in equation 3 is ***d,*** the head to medium spacing. This also figures heavily into the wavelength response upon read back. Loss of resolution of the shortest wavelengths is severe (e^{-kd} , where $\boldsymbol{k}$ is the wave number) and the resultant signal loss is normally given in dB form by the relation [6]

$$Loss = -54.6 \frac{d}{\lambda} \quad \mathbf{dB} \qquad \ldots (5)$$

In combination with spacing on write, the multiplier in equation 5 (-54.6) is closer to -100! Although we run the tape in physical contact with the head, the 'magnetic' spacing seen is due to media roughness, recession of the magnetic elements in the head and any

adherent (or temporary) debris or stains on the head. Current systems appear to have up to 70nm of magnetic spacing while in apparent physical contact and this will have to come down if we want to resolve high density terabyte recordings and not suffer the resulting loss in signal amplitude and resolution.

Head technology appears to have enough precedents and product introductions (again as seen in disk magnetic recording) that tape head offerings should be able to readily respond to new media types as they are developed. A classic example would be the shift to all thin film write heads and thin film shielded read heads as well as merged pole/shared shield structures commonly used in disk and now being seen in tape applications. Examples are shown in figure 5. The main issues facing the tape head

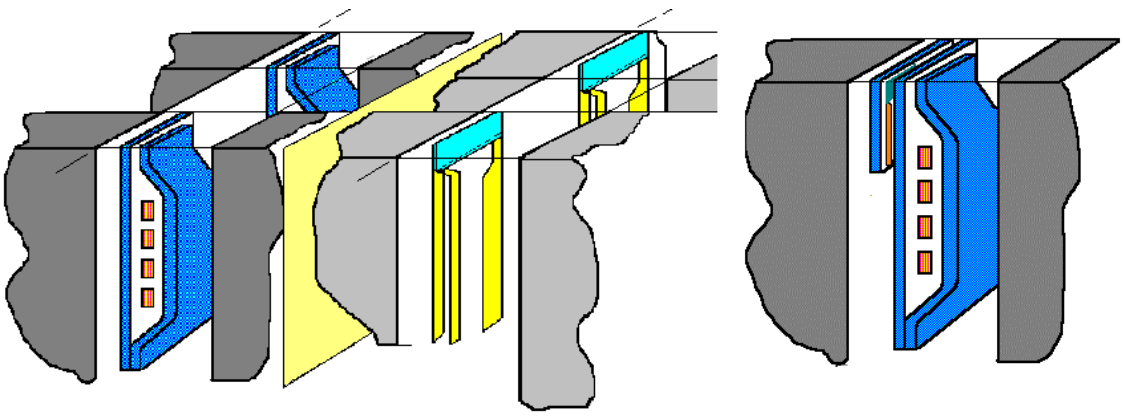


Figure 5. Diagrams of thin film tape head types showing thin film write, MR read and combination shared shield devices

concern the consequences of using multiple channels simultaneously in read-while-write mode. I.e., direct write to read feed through and read element off-track due to tape static and dynamic azimuth. A future example of disk like technology for tape would be the introduction of the GMR spin valve read sensor now prevalent in desktop systems in disk drives. This would be predicated by the availability of a suitable media that would be compatible with high-density recordings and these very sensitive devices, as well as environmental issues seen in tape usage. Alternatively, new designs of spin valve sensors customized for tape could be used with the still somewhat higher $M_r\delta$ values that may persist. The switch to spin valves will be driven by the need for raw signal amplitude to overcome the unique noise sources in the multi-channel read-while-write tape environment (such as write-to-read feedthrough) as the read element width and hence signal amplitude is reduced.

Another issue raised in Table 1 is the time scale of the recording. For high bpi and fast tape speeds, the bit cell time is reduced to <10nS. If write equalization persists as a favorable recording method (which it will if the $M_r\delta$ is not reduced significantly) then the recording system (write current, write head magnetics and media magnetization) will have to respond on the 1nS time scale. For a 3nS write equalization pulse the media has to see the field at least 2nS of that time to stand a chance of responding. Figure 6 shows how magnetic media (in this case MP1 media) changes its effective coercivity for fields

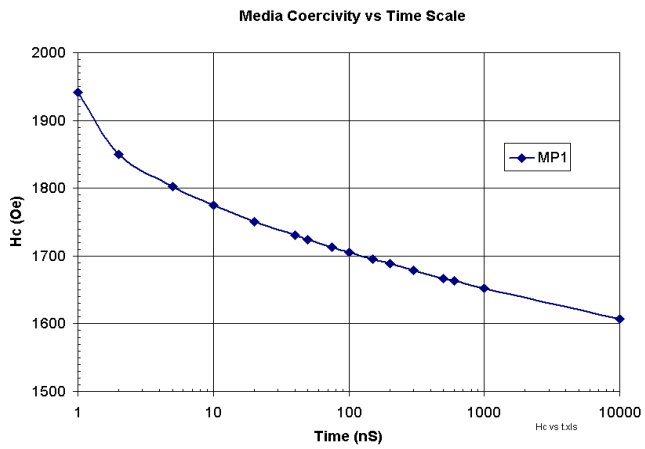


Figure 6. Coercivity of MP tape versus time scale of the applied magnetic field.

applied at very short times. The rise in coercivity means that we would have to overdrive the system to affect the recording in the required way presuming that the head magnetic core can provide the specified field in response to the drive current. The issue of getting the drive current into an inductive load like a write head exacerbates the problems in a multi-element tape head where stray capacitance paths can shunt the coil current. Core materials for the head appear to be available to provide this response and current production head types such as the StorageTek T9840B write head have demonstrated good performance down to 10nS. Data for this is shown in figure 7 for such a write head, which uses cobalt based amorphous alloy poles.

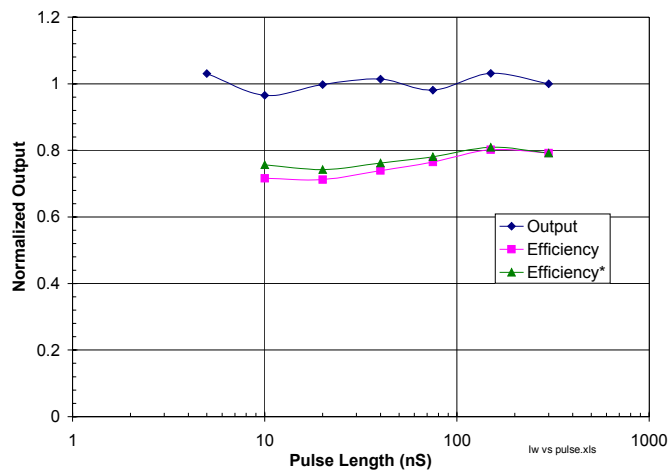


Figure 7. Head efficiency and readback output vs. write pulse length

This data shows that the read back amplitude remains the same when the pulse length is reduced and that the head efficiency does not roll off significantly. The two efficiency curves represent the directly measured head efficiency and the head efficiency corrected for the media coercivity shift according to figure 4 (this curve is indicated with an *). These time scale issues are not at any fundamental limits imposed by the laws of physics but provide the engineer with interesting challenges. The particulate and metal based tape media respond at 1nS and I think operating near 1nS will be avoided in any case with the eventual elimination of write equalization.

4. Mechanical Issues

The tape speeds used in Table 1, for the high data rates, provide the tape path and motion control with some challenges. This is especially so considering the tape lengths needed for the capacities which in turn means the tape thickness needed for the cartridge size

approaches 5µm (or even a little thinner). This thickness (or rather thinness) means a relatively low tape tension with which to achieve these speeds with adequate lateral guiding and tape pack management. On top of this, the bandwidth of a track following servo system would have to increase together with its capability to achieve the track pitch targets. Again, no real fundamental limits here, just a solid engineering problem. Unfortunately, these factors get much less attention than the more intuitive limits imposed on track density by the dimensional stability of the media itself. Very narrow tracks, coupled with multi-channel heads that span a significant portion of the width of the tape, result in track mis-registration (TMR) numbers that imply roadblocks beyond mere electronics. There is an interesting trade off between data rate and capacity that can be made as outlined in the 1998 NSIC tape roadmap [7]. Given a fixed tape length and achievable linear recording density, capacity can only be increased by increasing the track density (narrower tracks). This means the allowable off-track capability (OTC) is reduced. For higher data rates the only adjustable parameter, once the tape speed is set, is the number of parallel channels in the head. The more channels side by side the wider the span across the tape and more likely the end tracks will exceed the OTC as the tape dimensions change with time, tension, temperature and humidity. The results of the calculation of this trade off is formulated as

$$D = \frac{2(OT)LWVb^2\varepsilon^2}{64Cc_p m_c} \qquad \ldots (6)$$

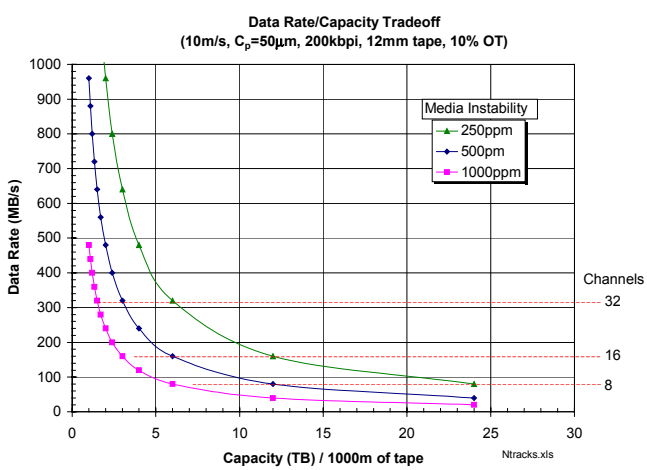


Figure 8. Data Rate/Capacity trade off for a linear tape system.

where ***OT*** is the allowable off-track expressed as a fraction of the track width, ***W*** the width of the tape, ***C*** the capacity of the cartridge, $\boldsymbol{c_p}$ the channel pitch in the head and $\boldsymbol{m_c}$ the media instability coefficient. Figure 8 shows the situation for various media stability numbers (from ref. 7) and is considered somewhat optimistic as it considers only writing the tracks in the correct location and not any read-while-write or realistic read back scenarios. Also current feedback from media suppliers is that the stability numbers will probably not improve as significantly as suggested here anytime soon. The implications of this chart are simple to interpret. If you want very high capacity, i.e., very narrow tracks, the number of parallel channels laid side by side will have to be reduced, lowering the possible data rate.

The only way around this is to change the way we parallel up head stacks to avoid the excessive head span or change in some other way we lay data on tape. Super DLT and helical systems for instance use dual azimuth recording on adjacent tracks allowing a larger OTC. This is one reason the areal density demonstrated by helical scan systems (e.g. SONY) already exceeds that projected for linear systems. Helical technology uses a single channel or few channels approach, high head-tape interface speeds, dual azimuth and short length tracks, which circumvent these media related problems. Unfortunately helical technology has suffered head and media wear problems and there is a perception of poorer reliability compared to linear systems, the basis for which is somewhat clouded. The challenge for the multi-channel linear head here is the reduced channel pitch. 50μm as indicated in Table 1 is certainly achievable, but beyond that, a new approach over

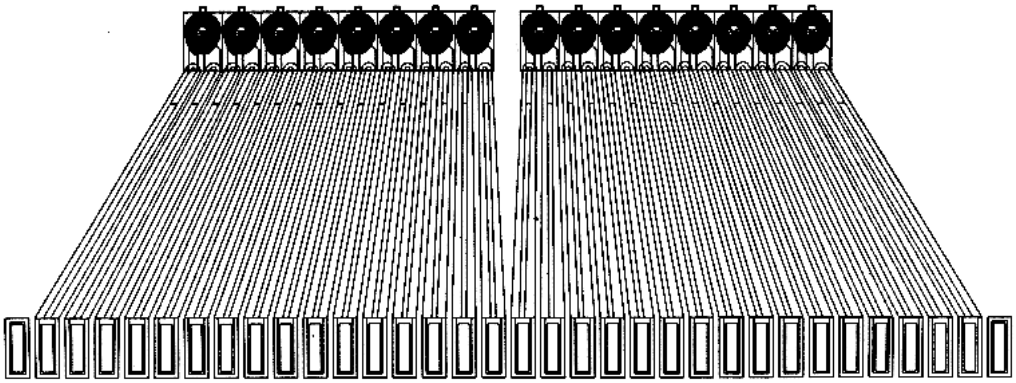


Figure 9. Example of a multi-channel, side-by-side architecture, thin film tape write head as used in today's linear tape heads [3]

today's norm (figure 9) is expected.

Finally, figure 10 summarizes the areal density progress and trend extension for linear tape based on past and present systems and published roadmaps. As mentioned before, heads and media in combination are the primary drivers for this parameter. The coercivity rise from oxide tapes to MP tapes and in the future thinly coated particulate or metal film tapes have been responded to by heads moving from ferrites to thin films and high moment thin films to write these tapes. This is in conjunction with MR and eventually GMR read heads to deliver appropriate signal quality. Also shown is the SONY helical 6.5 Gb/in^2 demonstration on metal evaporated (ME) tape and subsequent 16.4 Gb/in^2 point using spin valve heads [8], and the estimated MP limit using today's assumptions.

5. Conclusions

It is clear that the medium has a significant if not the primary impact on the density growth in magnetic tape recording. As demonstrated by disk magnetic recording the $M_r\delta$ has to be reduced in order to increase the linear density. Significant reduction in this parameter would allow closely spaced magnetic transitions and enable the use of more sensitive read head sensors such as spin valves to boost the sagging raw signal amplitude as both the bpi and tpi increase. Calculating a limit for MP tape throws down the gauntlet for media, head and channel developers to counter this, as was seen recently in magnetic

disk. There the areal density limit was calculated to be 36Gb/in^2 in 1997 [9], which is now exceeded in today's normal production disk drives!

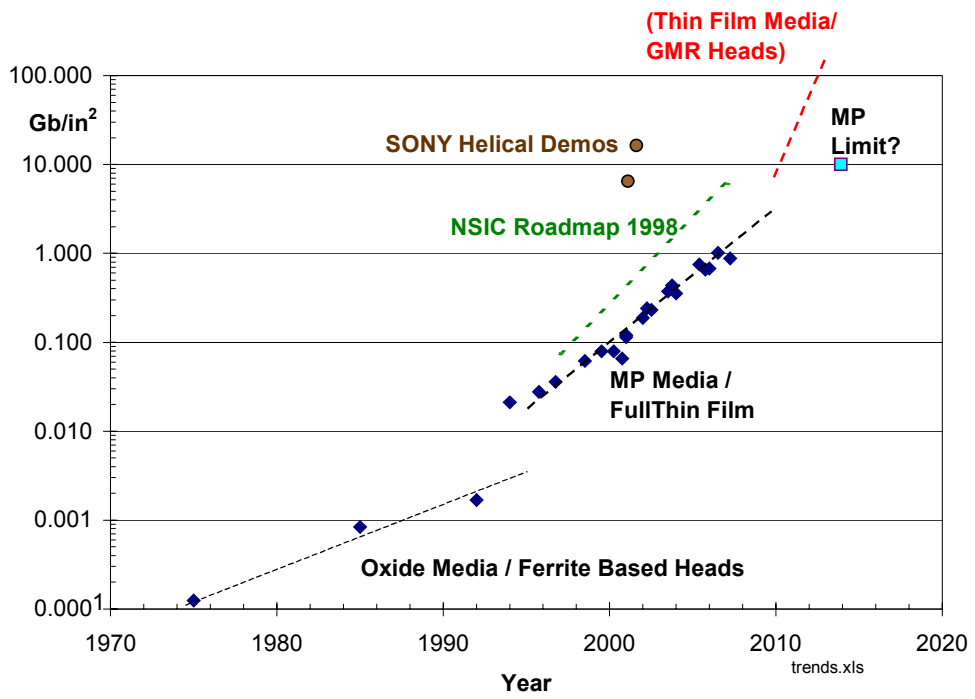


Figure 10. Areal density trends in linear tape systems

Increasing the data rate by increasing the number of parallel channels involves trade offs with tpi (i.e. capacity) if we remain with the side-by-side head stack architecture due to the increasing span of the active channels across flexible media, which is accepted as having somewhat poor dimensional stability. Head technology appears to be available to meet the challenge of the multi-terabyte capacity cartridge but this target is gated by media type and availability, and overcoming the engineering challenges of handling the magnetic and physical properties of the media. Tape is not nearing any fundamental scientific limits as seen in magnetic disk. Given the rather moderate areal densities currently seen in tape systems and optimism with regard to the development of tapes with thinner magnetic coatings, data storage systems using tape are poised to make some rapid advances in capacity *and* data rate.

6. References

[1] "Magnetic tape as the mass storage medium". T, Schwarz, Mass Storage Conference (2000).

[2] "The future of magnetic tape", T.Schwarz, Mass Storage Conference (2001).

[3] "Magnetic Tape Recording Technology and Devices", R. H. Dee, Proc. Int'l Non-Volatile Mem. Tech. Conf. pp55-64 (1998). (IEEE Cat. No. 98EX141).

[4] "The Foundations of Magnetic Recording", J. C. Mallinson, (Second Ed.), Academic Press (1993).

[5] "Investigation of particulate media with an ultra-thin magnetic layer suitable for MR heads on a rotating drum" K. Ejiri et al, IEEE Trans. Magn. Vol.37, No. 4, pp1605-1607 (2001) and Fuji Film press release on 'Nano Cubic Technology' www.fujifilm.com (2001).
[6] "The reproduction of magnetically recorded signals", R. L. Wallace Jr., Bell Syst. Tech J. vol.30, 1145 (1951). See also Ref. 2 p.88.
[7] "Tape Roadmap", National Storage Industry Consortium (June, 1998).
[8] "Beyond 6.5Gbit/inch^2 recording using spin valve heads in tape systems", T. Ozue et al (SONY), Proc. TMRC Conf., Minneapolis, MN (Aug, 2001).
[9] "Thermal stability of recorded information at high densities", S. Charap, P.Lu and Y.Lee, IEEE Trans. Magn. Vol. 33, No.1, pp. 978-983 (1997).

Efficient RAID Disk Scheduling on Smart Disks

Tai-Sheng Chang
tchang@cs.umn.edu
Tel: +1-847-856-8074
Department of Computer Science and Engineering,
University of Minnesota
200 Union Street SE #4-192
Minneapolis MN 55455

David H.C. Du
du@cs.umn.edu
Tel: +1-612-625-2560
Department of Computer Science and Engineering,
University of Minnesota
200 Union Street SE #4-192
Minneapolis MN 55455

1. Introduction

With the emerging high-performance storage systems as well as the availability of faster processors and high-speed networks, many applications that were only dreams a few years ago, have become reality. For example, Digital Libraries and Digital Medical Imaging Archive Systems have become available today. Many of these new applications are making great impacts on the way we work and the way we live. Among the supporting technologies, a high-performance storage system is one of the most critical factors in these systems.

RAID (Redundant Array of Independent Disks) has been playing a very important role in supporting high performance storage systems. It exists in storage systems ranging from one with a couple disks to those with several terabytes capacity. RAID uses data striping and parity information to provide higher I/O throughput on large data access and fault tolerance against disk failure. The implementation of RAID systems can be categorized into two different groups. The first category is the hardware RAID that uses additional RAID controllers to manage and process most of the required tasks in a RAID system. Those tasks include data parity computation and volume management. The other category of RAID uses the existing CPU(s) and memory on the system instead for all the necessary tasks (as opposed to the hardware RAID solution, we call it software RAID). From a user's point of view, hardware RAID solutions require RAID controllers and increase the costs of a system; On the other hand, Software RAID solutions consume CPU and memory resource when performing RAID operations. Therefore, the applications running on the same hosts where the software RAID resides will suffer performance degradation.

Fortunately, there is a new technology that provides an alternative solution between the expensive Hardware RAID solutions and the poorer performing Software RAID solutions. This new technology is called Disk-Based XOR. Disk-Based XOR is a technology utilizing the capability of computation on disks. By calculating the XOR results on disks, the CPU resource is no longer required for the computation-intensive XOR computation in RAID systems. Another big advantage of the Disk-Based XOR approaches is that the data amount needs to be transferred on storage channel can be greatly reduced by as much as 50%. With traditional RAID's, both old data and old parity

data have to be sent to the host or a RAID controller for new parity construction. The new data and the new parity will be then transferred back to the target data disk and parity disk, respectively. On the contrary, in a Disk-Based XOR RAID, only the new data and the XOR results of the new and old data will be transferred. Therefore, with Disk-Based XOR, up to twice as many disks could be connected to a storage channel without saturation under the similar load. This advantage has been proved with simulation results in an earlier study.

However, there are challenges in implementing a Disk-Based XOR RAID system. Because XOR calculations of the new and old data will be executed on the data disk and the results need to be transferred to the parity disk, the results have to be saved on data disk before the results have been transferred successfully to the parity disk. It may have a big impact on performance. Researchers have found a potential deadlock situation with traditional single-threaded executions of SCSI commands in Disk-Based XOR RAID's. Some researchers proposed a different RAID parity placement on disks to avoid such a problem. Another research showed the deadlock could be avoided with a small change on the FC-AL protocol. A multi-threaded SCSI command execution approach has been proposed not only to resolve the deadlock problem but also improve disk efficiency. The approach uses a conditionally prioritized disk command queue to resolve the deadlock problem. Simulation results were shown that such an approach outperformed a host-based RAID.

While the proposed multi-threaded XOR approach seems promising, it does raise another issue: The proposed conditionally prioritized disk command queue execution may conflict with disk scheduling discipline designed to optimize disk efficiency. The conflict is due to the fact that free cache segments may not be always available for the next new read-modify-write command. In such a case, one of the other commands will be executed next instead. As a result, a disk may not execute commands as efficiently as it could have been. In this paper, we will investigate the performance impact of such scheduling conflict and propose two new disk scheduling algorithms.

We choose a popular disk scheduling, Shortest Service Time First (or SSTF) as the base line for comparison. This method has been widely used and shown as having good performance in a dynamic environment where commands are arriving over time. In this paper, we call the SSTF scheduling a Greedy Algorithm. In this scheduling, each disk chooses the command with the shortest service time (seek time plus latency time) to be the next command. In the case when available cache segments are not enough for next read-modify-write operation, the command with the shortest service time among the other commands will be chosen. This is the same as in the proposed multi-threaded approach by other researchers in their study. The only difference is that in this paper, SSTF scheduling discipline will be used to choose from the list of executable commands. When no other commands are in the disk queue, a disk will be forced idle.

Two reasons may cause disk cache to build-up. The first is due to congested data links. When the disks are putting data to cache faster than cache can transfer data to the storage channel, the cache will be filled. This could happen when too many disks are connected to a single storage channel. This situation can be easily avoided with proper sizing when

configuring a system if the traffic load can be realized. In Disk-Based XOR, there is another possible cause. Disk cache segments filled with XOR results need to be protected until the associated parity update is completed. Depending on the disk scheduling discipline, a parity update command may take a long time waiting in disk queue before it has been executed. The longer the waiting time is, the longer time the associated cache segments on the target disk remains to be saved and protected from being used by other commands. Our proposed approaches will intend to reduce the waiting time of the parity updates.

The rest of this paper is organized as the following. In Section 2, we will provide a more detailed description of Disk-Based XOR operations. In Section 3, we will also describe in details the Greedy disk scheduling discipline and those two new enhancements. In Section 4, we will present our simulation results to show the performance of those three disk scheduling disciplines following an overview of our simulation model. Finally in Section 5, we will summarize what we found in this study and conclude the paper.

2. Disk-Based XOR and Its Operations

Three new SCSI commands (see [1]) have been created for supporting the Disk-Based XOR implementation. They are XD-write (or XDW), XP-write (or XPW), and XD-write extend (or XDW-ext). Each XDW is always associated with one XPW command. An XDW command consists of four operations. To begin, data (old data) will be read from target disk to its disk buffer (disk cache). At the same time, new data will be sending from the host to the target data disk. When both new and old data become available on disk buffer, exclusive-or operations will be executed on the new and old data. The new data will be written onto the disk. The results of the XOR operations, on the other hand, will remain on the disk buffer for later use by the associated XPW. The results need to be saved and protected on the disk buffer from being overwritten by other operations. Figure 1 shows an XDW operation.

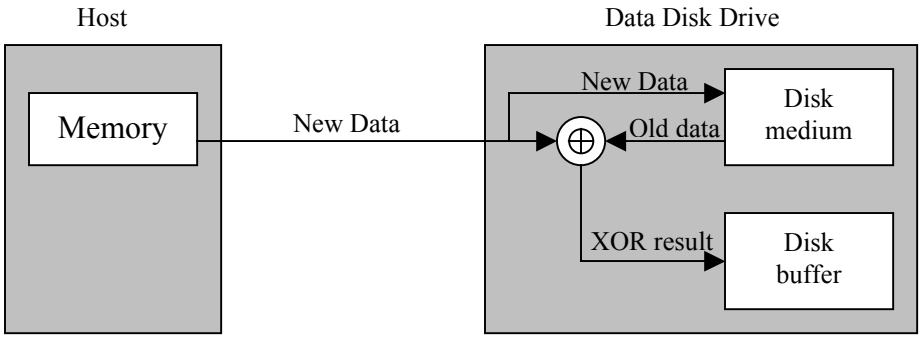


Figure 1: XDW Operation

After an XDW command is completed, the associated XPW command will be sent to the associated parity disk. The old parity will be read from the disk medium. At the same time, the XOR results of the associated XDW command stored earlier on the target data disk will be sent to the parity disk. When the XOR results and old parity information

become available, XOR operations will be executed. The newly derived XOR results will be written onto the parity disk. After the XPW has completed, the disk buffer storing the XOR results saved on the target data disk by the associated XDW will be freed. Figure 2 shows the operations of an XPW command.

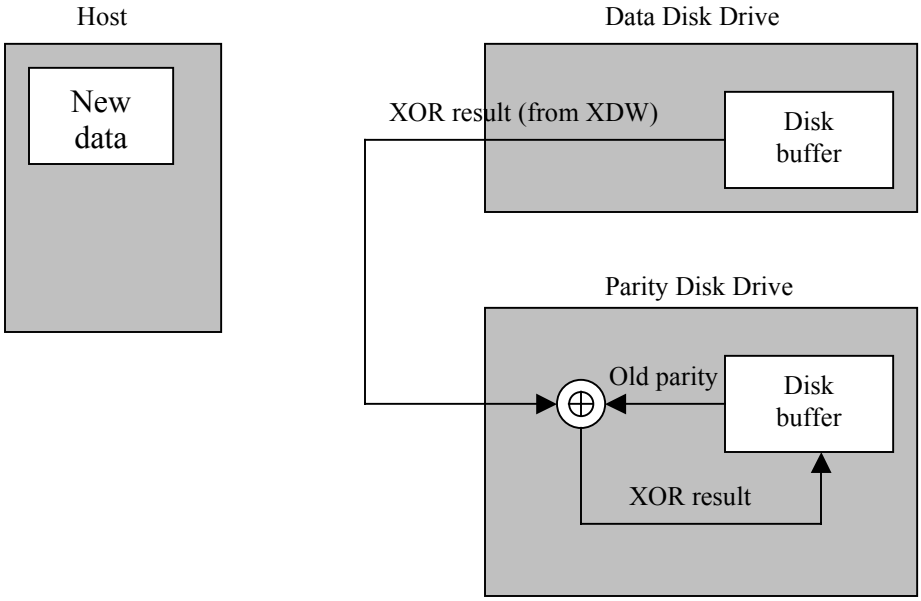


Figure 2: An XPW Operation

An XDW-ext command is a macro command that consists of one or more XDW commands followed by the associated XPW command(s). A read-modify-write operation on a data block can be fulfilled by an XDW-ext command.

One big advantage of the Disk-Based XOR approach is that the data amount being transferred on storage channel can be greatly reduced by as much as 50%. With the traditional RAID's (either hardware or software RAID's), both old data and old parity data have to be firstly sent to the host or a RAID controller to construct the new parity data. The new data and the newly derived parity data will be transferred back to the target data disk and parity disk, respectively. In other words, if we need to update a block of data, there will be four blocks of data that are required to be transferred from and to the disks. As opposed to the traditional RAID's, in a Disk-Based XOR RAID it only needs to transfer the new data and the XOR results of the XDW on the storage channel. Therefore, with Disk-Based XOR, a larger number of disks can be connected to a storage channel before saturating it with the same disk load.

3. Two XPW-Enhanced Disk Scheduling Disciplines

Many disk scheduling disciplines have been proposed to improve disk efficiency. For example, SCAN and C-SCAN ([2]) were proposed to reduce the seek time without moving back and forth from one request to another. Some other approaches considered to reduce both seek time and rotation latency (i.e. disk service time). Shortest Service Time

First (SSTF) is one of those approaches and has been widely used as the disk scheduling discipline.

Before RAID was first introduced, disks operated individually and independently. There was no correlation between any two operations on different disks in terms of their access location on disks. RAID changed such independency. Updating a data block on one disk in a RAID will result in updating the associated parity block that has the same Logical Block Address (LBA) as the data blocks but resides on a different disk (parity disk). However, most disks in a RAID (except RAID-3) are still operating independently without coordination between disks. That is, reading the old data from a disk is performed independently with the reading of the associated old parity data from another disk. Because the new parity data is constructed by the old data, old parity data and the new data, intermediate results must be saved before both the old data and old parity are available. Without collaboration, the retrievals of the old data and old parity will be scheduled independently on two disks. As a result, the intermediate results may have to be saved for a long period of time. That is why most of the RAID systems require a large amount of memory either on the RAID controller or on the host.

Such a big memory requirement is impractical in a Disk-Based XOR RAID. With a very limited buffer space on most disks, disk buffer can be filled quickly with Disk-Based XOR operations. When the disk buffer is full, no more commands will be executed until some buffer becomes available. A more severe condition is that a deadlock may happen when the buffer is full in Disk-Based XOR. That is why in [3], the proposed conditional prioritized disk scheduling forced a disk to choose a command other than XDW-ext after the occupancy of the disk buffer is higher than a predefined threshold. However, such an alternation on the disk scheduling will have an impact on the disk efficiency. The disk efficiency could be much lower when choosing a sub-optimal command.

In the following, we will introduce two XDW-enhanced algorithms. Both of them are intended to reduce the probability of being required to make a dramatic change on disk scheduling. As for a baseline comparison, we use a greedy algorithm with the SSTF scheduling. The discussion of this Greedy scheduling approach is also included in the following sections.

3.1 Greedy Disk Scheduling

The Greedy algorithm chooses the command with the shortest service time (seek time plus latency time) to be the next command to be executed. This method has been widely used and performs well in dynamic environment where commands are arriving over time. We use this method as a baseline for comparison purpose.

Because cache may be filled in Disk-Base XOR as discussed in the previous section, some modification is needed when applying the Greedy method to Disk-Based XOR RAID's. Each XDW-ext command requires at least two segments of cache to store data; one for the old data from disk and another for the new data from the host (assuming request data size is less than or equal to the segment size). Hence, we need at least two

segments of free cache space in order to start execution of an XDW-ext command. When the number of available cache segments is small enough for the next XPW-ext command, we change the Greedy Algorithm and choose the command with the shortest service time from commands other than XDW-ext commands. The modified greedy method is used in this paper as a performance baseline to compare with the proposed (two) enhancements.

As discussed in the previous section, one drawback of the Greedy method in Disk-Based XOR is that when it is running out of free cache space, it has to pick a sub-optimal command, or even worse, stay idle. In a case when there is no command other than XDW-ext in the disk queue, the disk has to stay idle until either a new non-XDW-ext command arrives or some cache space is freed.

One straight forward way to reduce such inefficiency is to prevent it from happening. There are two reasons causing the cache to back up. The first is due to a congested link. When the disks are putting data to cache faster than cache can transfer data to the storage channel, the cache will be filled. This could happen when too many disks are connected to a single storage channel. This problem may be eliminated with proper system sizing when configuring a system.

In Disk-Base XOR, there is another possibility. That is when the number of outstanding XDW-ext commands on a disk is close to the number of cache segments. An outstanding XDW-ext command is an XDW-ext command finishing its XDW part but waiting for its XPW part to be complete on another disk. Depending on the disk scheduling discipline, an XPW command may take a long time waiting in disk queue before it is executed. The longer the wait time, the longer the cache segment on the data disk needs to be saved and protected from being used by other commands.

After understanding the cause of a long-waiting outstanding XDW-ext command, we proposed two approaches to reduce the possibility of filled cache in Disk-Based XOR RAID's. The details are in the next two subsections.

3.2 An XPW Service Time Based Promotion Scheme (XPWT)

The first approach is to selectively give an XPW the higher priority. By giving XPW commands higher priority, it helps to reduce its wait time in disk queue and as a result, the associated XDW-ext command can be completed and release the cache space it used earlier. However, selecting XPW should be made with caution such that the disk efficiency will not be over-compromised. We use a relative difference in disk service time as the criteria to give an XPW the higher priority. When an XPW has less than smallest service time plus the predetermined time δ available, the XPW with the smallest service time will be given the highest priority and will be executed next.

We formulate the approach proposed above in the following.

Let C_{All}^{min} be the command with the shortest service time T_{All}^{min}
Let C_{XPW}^{min} be the XPW command with the shortest service time among XPW commands T_{XPW}^{min}.

If $T_{XPW}^{min} - T_{All}^{min} <= \delta$ then choose C_{XPW}^{min} to be the next command.
Otherwise choose C_{All}^{min}.

Note that when δ equal to zero, this approach degenerates to the Greedy Algorithm. On the other hand, when δ becomes a large number, XPW commands will be given the higher priority all the time. For example, when δ is greater than or equal to the largest possible disk service time, the above method will always give the higher priority to XPW commands.

3.3 An XPW Queue Length Based Promotion Scheme (XPWQ)

The performance of the previous approach highly depends on the value of δ. Choosing a large δ may result in lower disk efficiency but reduce the number of XPW's in disk queue; while choosing a small δ makes it closer to the Greedy Algorithm. Therefore, the optimal value of δ is difficult to determine in a dynamic situation. The second approach we are proposing in this paper is to give XPW commands the higher priority when the number of XPW commands on a disk reaches a certain threshold. The idea is based on the fact that with a uniformly distributed access among disks in a RAID and a large number of XPW commands in one disk queue, the more occupied disk cache will be on the other disks. Therefore, choosing an XPW to execute will likely help in releasing the disk cache buffer on another disk. Furthermore, when the threshold is chosen properly, there will be a set of XPW commands in disk queue to choose from when the number of occupied cache segments reaches the threshold. The larger the number of XPW commands to choose from, the closer the chosen XPW command to the optimal command. The detailed formulation of this approach is provided in the following.

Let MaxNxpw be the threshold value of the number of XPW commands.
Let Nxpw be the number of XPW commands in a disk command queue.

If Nxpw <= maxNxpw then follow the Greedy Algorithm.
Otherwise, pick the XPW command with the shortest service time of all XPW's.

Note that when the value of maxNxpw is set to zero, this approach will always choose an XPW if one exists. On the other hand, when the value of maxNxpw is set to infinity, then this approach will not give XPW a special higher priority at any case. Therefore it will degenerate to the Greedy Method.

4. Simulation Model and Results

In this section, we will use simulation results to demonstrate the performance difference of the three disk-scheduling disciplines discussed in the previous section. For better understanding of the simulation results, we first provide an overview of our simulation models in the following subsection.

4.1 Simulation Model

We used a storage subsystem simulation model to simulate operations of a storage subsystem based on the Fibre Channel - Arbitration Loop (FC-AL) ([5]) protocol. The model consists of three major components: A disk and its disk cache component; A storage interface component that follows FC-AL protocol and controls data transfers to/from the storage channel; And a command generator component that simulates a host generating data requests.

4.1.1 Disk and Disk Cache Model

The disk model is based on an IBM Ultrastar XP 4.51GB disk. The implementation of this disk model employs zone bit recording and non-linear seek time functions for read and write operations using information from the disk manufacture in [6]. Table 1 shows a summary of disk parameters used in the simulation.

Table 1: Disk Parameters

Disk Parameters	Value
Capacity	4.51 GB
Rotation Speed	7202.7 RPM
Average rotation latency	4.17 ms
Seek times	0.5 – 16.5 ms
Transfer rate	5.53 – 7.48 MB/sec

Disk cache is the buffer for temporarily storing data sent to/from the storage interface. It is partitioned into segments. Each segment consists of many 512-byte blocks. In our simulation model, each segment will be used by one command. The cache component also employs an LRU (Least Recently Used) cache segment replacement scheme. The parameters that the disk cache used in the model are summarized in Table 2. In our simulation, the number of segments is a controlled parameter. We used different numbers of segments in order to understand the impact of cache size and disk scheduling schemes on the system performance.

Table 2: Disk cache parameters

Disk Cache Parameter	Values
Block Size	512 bytes
Number of segments	Varied
Segment size	64 KB

4.1.2 FC-AL Model

We follow the FC-AL standard to model our disk interface. FC-AL is a protocol allowing Fibre Channel to operate in a loop topology. It is logically located between FC-1 and FC-2. The FC-AL component in our model consists of both Loop Port State Machine (LPSM) and Fibre Channel Protocol for SCSI (FCP). LPSM defines the behavior of the FC-AL loop port. It includes an arbitration protocol which determines who can access the loop. It also includes a fairness protocol that enforces fair sharing of loop among all the nodes. FCP is one of the Fibre Channel mapping protocols (FC-4) which uses the service provided by FC-PH to transmit SCSI commands and data. It also transmits status information between a SCSI initiator and a SCSI target. More details about FC-AL can be found in [5] and [7]. Table 3 summarizes the parameters we used in the FC-AL model.

Table 3: FC-AL Simulation parameters

FC-AL Simulation Parameters	Values	Descriptions
Link Speed	100 MB/Sec	Bandwidth of an FC-AL loop
Propagation Delay	3.5 ns	Propagation delay between two nodes
Per Node delay	6 word time	The delay of forwarding a frame by interface
Fairness algorithm	Enabled	The fairness protocol in its arbitration scheme

4.1.3 Command Generator

Command Generator is responsible for generating commands in our model. At the beginning of each simulation run, it will generate the number of commands indicated by the value of the maximum outstanding command parameter. When a command finishes, it will generate another command immediately to maintain the maximum outstanding commands in the system. The target disk of each command and the command's access location (LBA) on the disk will be randomly assigned by the command generator. The Command generator is also responsible for sending the SCSI command response to the target disk and generating data to be written on disks.

4.2 Simulation Results

To better understand the impact of disk scheduling on Disk-Based XOR, we conducted simulations in many different scenarios. We compared three disk scheduling disciplines under different system loads with different data request sizes. We also compared them in small and large-scale storage systems. To predict the impact of the three different disk scheduling algorithms on the Disk-Based XOR RAID performance with the high-end disks, we further conducted simulations using a disk model with a two times improvement in the disk rotation and seek times. By conducting these different simulations, we hope to provide a better view of the impact of the disk scheduling on Disk-Based XOR RAID performance and therefore, to demonstrate its importance.

To better present our results, we will use an eight-disk FC-AL model as a base model. We will compare the performance by changing the system parameters such as system load, data request size, and number of disks while keeping the other parameters the same. As the base model, We will show the average command response time for 4KB read-modify-write requests in the eight-disk FC-AL system. The total number of outstanding commands was 768. That is, the number of outstanding commands was maintained at 768 after the simulation started. A new command was generated immediately after a previous command had completed. For the XPWT scheduling, the value δ was set to 3 milliseconds. That is, an XPW command was given the higher priority over XDW commands if its disk service time is less than the smallest service time among all the XDW commands plus 3 milli-seconds. The maxNxpw value was set to the number of segments minus four. That is, the XPW commands in disk command queue will be given a higher priority when the total number of XPW commands in that disk command queue is greater than the number of cache segments minus four. For example, if the number of cache segments is twelve and there are more than eight XPW commands in disk queue, the next command will be chosen from those XPW commands in the queue. In such a case, the XPW with the shortest service time among the XPW commands will be chosen as the next command.

The simulation result of the base model is shown in Figure 3. The XPWT Algorithm has the least average command response time among the three on all the cache segment sizes used in this study. It was 7% better than the Greedy algorithm when the number of segments is eight. The results of the XPWQ Algorithm varied with the number of segments. When the number of segments was eight, it performed closely to the XPWT Algorithm. When the number of segments increases, the response time fell between those of the Greedy Algorithm and the XPWT Algorithm.

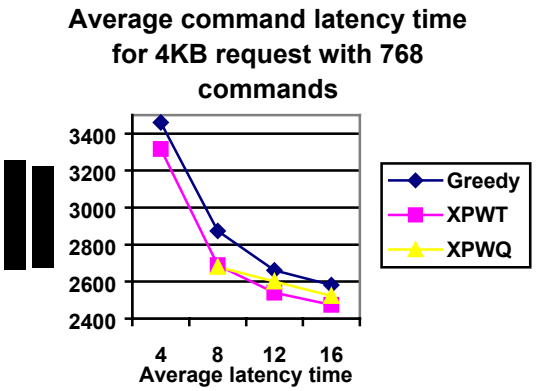


Figure 3:Average command latency with 4KB requests and 768 outstanding commands.

Figure 4 shows the system throughput achieved by the three scheduling algorithms on the base model. Since the system was loaded with a fixed number of outstanding commands (768 commands), the throughput was highly dependent on disk efficiency. The more efficient the disk is, the higher throughput it will generate. In Figure 4, we see that the XPWT Scheduling had the highest throughput among the three methods and had about 7% higher throughput than that of the Greedy Method in certain cases.

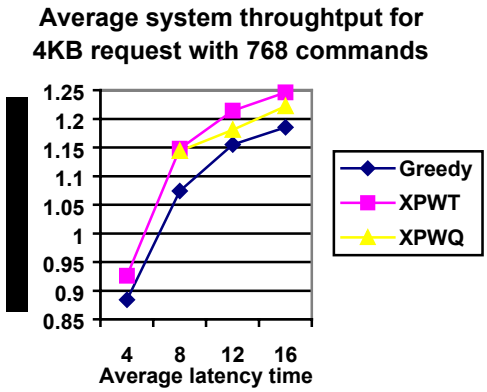


Figure 4:Average system throughput with 4KB requests and 768 outstanding commands.

Different System Loads

To understand the impact of the three different scheduling methods under different levels of the system loads, we also investigated the performance difference with a different number of outstanding commands in the system. As opposed to 768 outstanding commands, we conducted simulations with 512 outstanding commands on the 8-disk model. Figure 5 shows the results with both 768 and 512 outstanding commands. With 512 outstanding commands, the average command latency time was about two thirds of the time with 768 commands. The XPWT method outperformed the other two with 512 outstanding commands in all the three numbers of segments. The difference between the Greedy Method and XPWT Method was reduced from about 7% with 768 outstanding commands to about 5.4% with 512 outstanding commands. From the results, we found that the larger the number of outstanding commands, the higher the performance gap is between the XPWT method and Greedy Method. The major reason is that with more outstanding commands, it is more likely to execute an XDW command than an XPW command. When the cache segments are all filled, the disk will be forced to execute an XPW command. In such a case, the efficiency of the disk will be compromised.

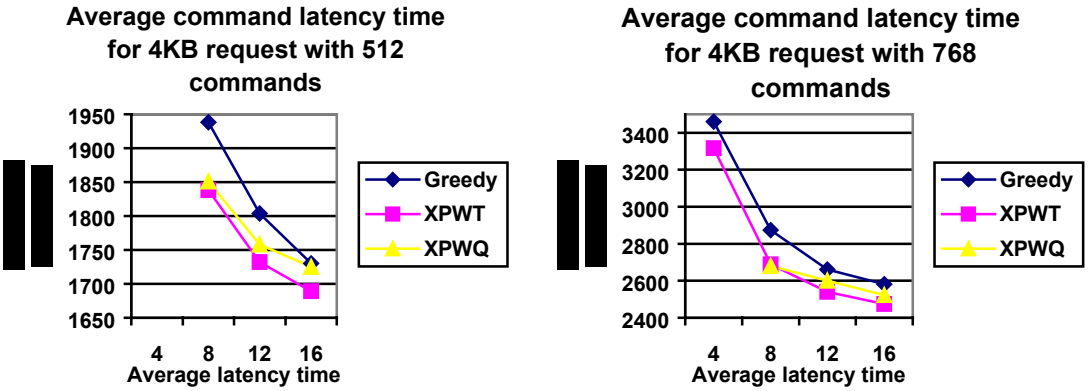


Figure 5: Average command latency with and 512 vs. 768 outstanding commands with 4KB requests.

Another observation from the results is that the XPWQ method tended to be close to the performance of the XPWT Method when the number of segments is small. On the other hand, it tended to be close to the Greedy Method's performance when the number of segments is large. This is because when the number of segments is large, more XPW commands are allowed in a disk queue before they are given the higher priority. Therefore, most of the time, the XPWQ method may perform as the Greedy Method. While with a smaller number of segments, it is more likely to reach the maxNxpw threshold. Therefore, it performs closer to the XPWT Method.

Large Scale Disk System

We conducted simulations on a 32-disk FC-AL model to show the performance in a system with a larger number of disks. In order to eliminate the performance difference resulted from disk queuing time between the eight-disk and 32-disk model, we used the same system load on both systems. We used an average of 64 commands per disk. That is, we used 512 outstanding commands on the eight-disk model and 2048 commands on the 32-disk model. The results showed a similar trend to what we have observed in the eight-disk model (See Figure 6). The XPWT Method was still the best among the three. It is about 7% better than the Greedy Method when the number of segments was equal to eight. The XPWQ Method performed just as well as the XPWT Method when the number of segments was equal to eight. But the XPWT method outperformed the XPWQ method when the number of segments became larger.

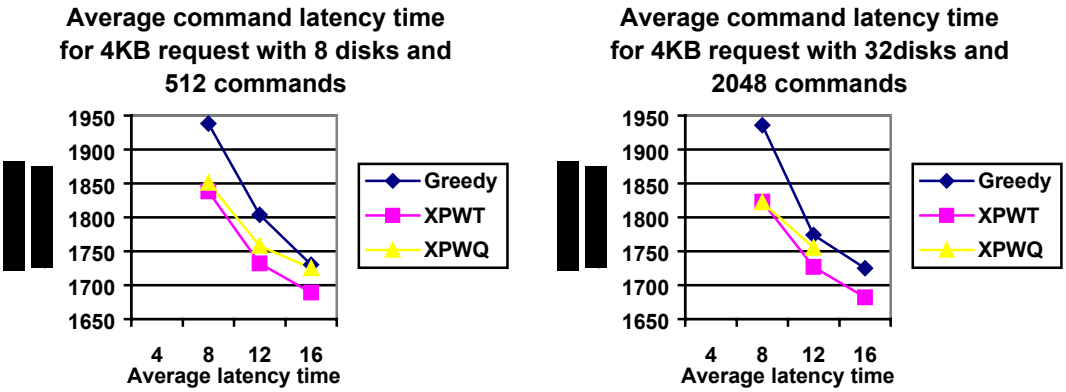


Figure 6: Average command latency with 8 vs. 32 disks with 4KB requests.

Large Request Size - 64KB:

With a 4 KB request size, the actual transfer time is less significant compared to the disk seek time and latency time. Therefore, the disk scheduling has a greater impact on the disk efficiency. As the request size increases, the data transfer time becomes larger. The extent of the improvement with better disk scheduling may be different. To understand the performance of the three disk scheduling disciplines with larger requests, we also conducted simulations with 64 KB requests. The results are shown in Figure 7.

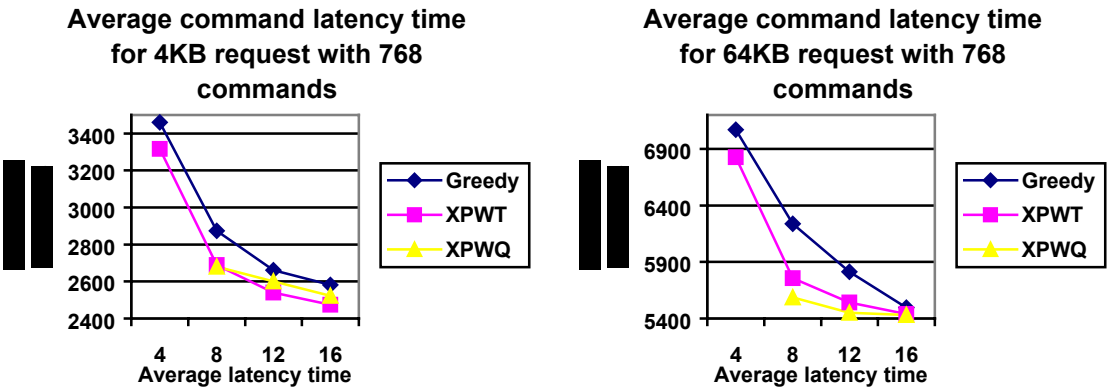


Figure 7: Average command latency with 4KB vs. 64KB.

With 64 KB requests, we observed even better improvement than 4 KB requests with XOR-enhanced scheduling when the number of segments is small. For example, with 4 KB requests, the improvement of the XPWT Method over the Greedy Method was about 7% with 8-segment cache. While with 64 KB requests, the improvement was more than 8%. Furthermore, the XPWQ Method outperformed both the other methods and had an improvement of close to 12% over the Greedy Method with an 8-segment cache.

Performance with the Faster Disks

Disk technologies have improved significantly over the past decades. Recently, disk density has been doubling better than every couple years. The disk rotation speed and seek time have also improved significantly. In this paper, we have compared the performance comparison of different disk scheduling disciplines with disk rotation speed that is used by most of the current off-the-shelf disk products (at the time this paper was written). To predict their performance with the faster disk speed, we also conducted simulation with faster disks.

In order to reuse our disk model and its very detailed seed functions and zone-bit encoding, we modeled the next generation disks by changing the parameters in our existing disk model. With the targeted 15000 RPM next generation disk, we believe that by doubling the disk rotation speed and halving the seek time and data transfer time in the disk model we have, it will give us a close approximation of the model for the next high-end disk. Figure 8 shows the performance comparison of the three scheduling methods with current and high-end disk models. The result is shown in Figure 8. The improvement of the XPWT method is almost 10% better than the Greedy method. The improvement of the XPWQ method fell between the Greedy method and XPWT method. It has about a 6.7% improvement over the Greedy method at eight segments.

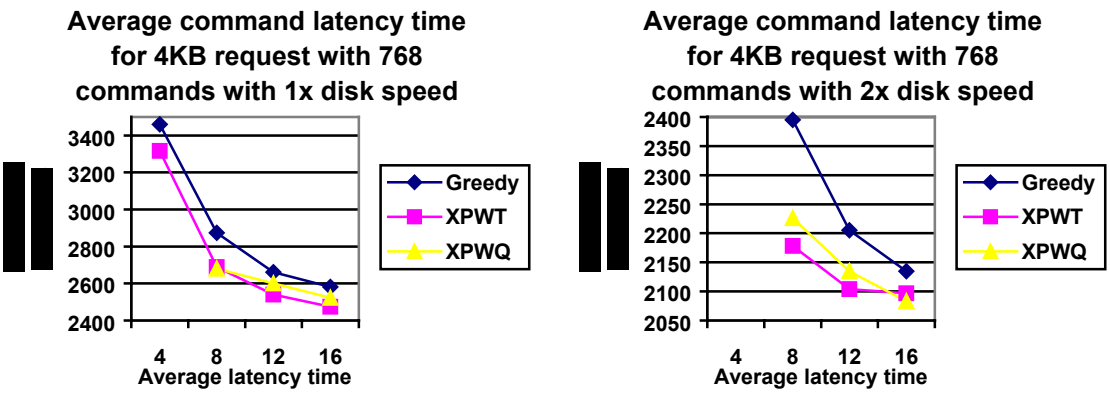


Figure 8: Average command latency with 1x vs. 2x disk speed with 4KB requests.

Impact of δ value in XPWT method

In the earlier section, we mentioned that choosing a good δ in XPWT could be difficult. To understand the impact of δ on the performance, we conducted more simulations with different δ values in different loads and cache segments. Figure 9 shows the results of the average latency when δ changes. The results show that when the number of outstanding commands is 768 and the number of segments is four, we should use a greater δ value. When the number of outstanding commands is 512, the optimal value falls when δ is around three to four. The results also demonstrate that when the number of segments is small, δ should be set to a greater value. In Figure 9, it seems that setting δ to 3 could provide a performance gain close to optimal except when the number of outstanding commands is 768 and the number of cache segments is four.

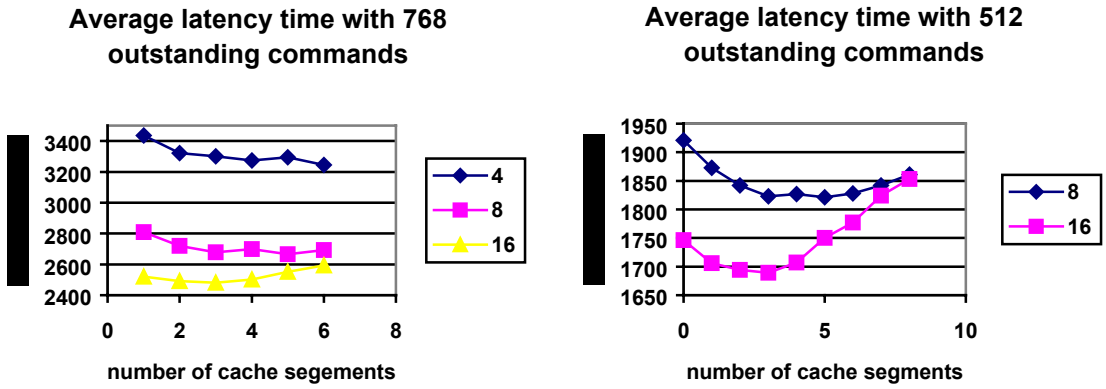


Figure 9: Average command latency with 1x vs. 2x disk speed with 4KB requests.

5. Conclusion

In this paper, we have discussed the uniqueness of Disk-Based XOR operations on disk scheduling and its impact on disk efficiency. We have proposed two XPW-enhanced disk scheduling disciplines that are designed to improve the disk efficiency on Disk-Based

XOR RAID's. We have demonstrated their performance results by simulations. We have investigated the performance of the proposed XPW-enhanced disk scheduling as well as the SSTF approach serving as the baseline performance. We have conducted simulations under different scenarios such as different scales of storage system, different system loads, different request sizes, and even with high-end disk technologies. We have demonstrated using simulation results that the performance was consistently improved with those two XPW-enhanced approaches throughout all the cases. The results showed that the improvement could be as much as 12%.

As the disk technologies continue to improve rapidly, it has been predicted that a one terabyte disk costing below one hundred dollars could be on the market in less than five years. With the price of disk going lower and lower, and the capacity of disks going higher and higher, it becomes more important to have a better RAID solution. Disk-Based XOR provides a promising lower-cost high-performance alternative. We hope that the study we have presented in this paper could open a door to finding better RAID solutions.

References

[1] Gerry Houlder, Jay Elrod, and Mike Miller, "*XOR Commands on SCSI Disk Drives*", X3T10/94-111r9.

[2] Avi Silberschatz and Peter Galvin, "*Operating System Concepts*", Addition-Wesley Publishing Company, Inc. fourth Edition, 1995.

[3] Sangyup Shim, Yuewei Wang, Jenwei Hsieh, Tai-Sheng Chang, and David H.C. Du, "*Efficient Implementation of RAID-5 Using Disk Based Read Modify Writes*" Technical Report, Department of Computer Science, University of Minnesota, 1996.

[4] Tai-Sheng Chang, Sangyup Shim, and David H.C. Du, "*The Designs of RAID with XOR Engines on Disks for Mass Storage Systems*", Sixth NASA Goddard Conference on Mass Storage Systems and Technologies in Cooperation with the Fifteenth IEEE Symposium on Mass Storage Systems, March 22- 24, 1998, College Park, Maryland.}

[5] David H.C. Du, Tai-Sheng Chang, Jenwei Hsieh, Yuewei Wang and Simon Shim. "*Emerging Serial Storage Interfaces: Serial Storage Architecture (SSA) and Fibre Channel - Arbitrated Loop (FC-AL)*", TR 96-073, Technical Report, Department of Computer Science, University of Minnesota}

[6] IBM Corporation, "*Functional Specification, Ultrastar XP Models*", 1995.

[7] David H.C. Du, Jenwei Hsieh, Tai-Sheng Chang, Yuewei Wang and Simon Shim, "*Performance Study of Serial Storage Architecture (SSA) and Fibre Channel - Arbitrated Loop (FC-AL)*", to appear in IEEE Concurrency

Experimentally Evaluating In-Place Delta Reconstruction

Randal Burns
Dept. of Computer Science
Johns Hopkins Univ.
randal@cs.jhu.edu

Larry Stockmeyer
Dept. of Computer Science
IBM Almaden Research Center
stock@almaden.ibm.com

Darrell D. E. Long
Dept. of Computer Science
Univ. of California, Santa Cruz
darrell@cs.ucsc.edu

Abstract

In-place reconstruction of delta compressed data allows information on devices with limited storage capability to be updated efficiently over low-bandwidth channels. Delta compression encodes a version of data compactly as a small set of changes from a previous version. Transmitting updates to data as delta versions saves both time and bandwidth. In-place reconstruction rebuilds the new version of the data in the storage or memory space the current version occupies – no additional scratch space is needed. By combining these technologies, we support large-scale, highly-mobile applications on inexpensive hardware.

We present an experimental study of in-place reconstruction algorithms. We take a data-driven approach to determine important performance features, classifying files distributed on the Internet based on their in-place properties, and exploring the scaling relationship between files and data structures used by in-place algorithms. We conclude that in-place algorithms are I/O bound and that the performance of algorithms is most sensitive to the size of inputs and outputs, rather than asymptotic bounds.

1 Introduction

We develop algorithms for data distribution and version management to be used for highly-mobile and resource-limited computers over low-bandwidth networks. The software infrastructure for Internet-scale file sharing is not suitable for this class of applications, because it makes demands for network bandwidth and storage/memory space that many small computers and devices cannot meet.

While file sharing is proving to be the new prominent application for the Internet, it is limited in that data are not writable nor are versions managed. The many recent commercial and freely available systems underscore this point, examples include Freenet [1] and GnuTella [2]. Writable replicas greatly increase the complexity of file sharing – problems include update propagation and version control.

Delta compression has proved a valuable tool for managing versions and propagating updates in distributed systems and should provide the same benefits for Internet file sharing. Delta-compression has been used to reduce latency and network bandwidth for Web serving [4, 20] and backup and restore [6].

Our in-place reconstruction technology addresses one of delta compression's major shortcomings. Delta compression makes memory and storage demands that are not reasonable for low-cost,

low-resource devices and small computers. In-place reconstruction allows a version to be updated by a delta in the memory or storage that it currently occupies; reconstruction needs no additional scratch space or space for a second copy. An in-place reconstructible delta file is a permutation and modification of the original delta file. This conversion comes with a small compression penalty. In-place reconstruction brings the latency and bandwidth benefits of delta compression to the space-constrained, mass-produced devices that need them the most, such as personal digital assistants, cellular phones, and wireless handhelds.

A distributed inventory management system based on mobile-handheld devices is an archetypal application for in-place technology. Many limited-capacity devices track quantities throughout an enterprise. To reduce latency, these devices cache portions of the database for read-only and update queries. Each device maintains a radio link to update its cache and run a consistency protocol. In-place reconstruction allows the devices to keep their copies of data consistent using delta compression without requiring scratch space, thereby increasing the cache utilization at target devices. Any available scratch space can be used to reduce compression loss, but no scratch space is required for correct operation. We observe that in-place reconstruction applies to both structured data (databases) and unstructured data (files), because they manipulate a delta encoding, as opposed to the original data. While algorithms for delta compressing structured data are different [9], they employ encodings that are suitable for in-place techniques.

1.1 Delta Compression and In-Place Reconstruction

Recent developments in portable computing and computing appliances have resulted in a proliferation of small network attached computing devices. These include personal digital assistants (PDAs), Internet set-top boxes, network computers, control devices, and cellular devices. The data contents of these devices are often updated by transmitting the new version over a network. However, low bandwidth channels and heavy Internet traffic often makes the time to perform software update prohibitive.

Differential or delta compression [3, 13, 9, 8], encoding a new version of a file compactly as a set of changes from a previous version, reduces the size of the transmitted file and, consequently, the time to perform software update. Currently, decompressing delta encoded files requires scratch space, additional disk or memory storage, used to hold a second copy of the file. Two copies of the file must be available concurrently, as the delta file reads data from the old file version while materializing the new file version in another region of storage. This presents a problem because network attached devices often cannot store two file versions at the same time. Furthermore, adding storage to network attached devices is not viable, because keeping these devices simple limits their production costs.

We modify delta encoded files so that they are suitable for reconstructing the new version of the file *in-place*, materializing the new version in the same memory or storage space that the previous version occupies. A delta file encodes a sequence of instructions, or *commands*, for a computer to materialize a new file version in the presence of a *reference* version, the old version of the file. When rebuilding a version encoded by a delta file, data are both copied from the reference version to the new version and added explicitly when portions of the new version do not appear in the reference version.

If we were to attempt naively to reconstruct an arbitrary delta file in-place, the resulting output

would often be corrupt. This occurs when the delta encoding instructs the computer to copy data from a file region where new file data has already been written. The data the algorithms reads have already been altered and the algorithm rebuilds an incorrect file.

We present a graph-theoretic algorithm for modifying delta files that detects situations where a delta file attempts to read from an already written region and permutes the order that the algorithm applies commands in a delta file to reduce the occurrence of conflicts. The algorithm eliminates the remaining conflicts by removing commands that copy data and adding explicitly these data to the delta file. Eliminating data copied between versions increases the size of the delta encoding but allows the algorithm to output an in-place reconstructible delta file.

Experimental results verify the viability and efficiency of modifying delta files for in-place reconstruction. Our findings indicate that our algorithm exchanges a small amount of compression for in-place reconstructibility.

Experiments also reveal an interesting property of these algorithms that conflicts with algorithmic analysis. We show in-place reconstruction algorithms to be I/O bound. In practice, the most important performance factor is the output size of the delta file. This means that heuristics for eliminating data conflicts that minimize lost compression are superior to more time efficient heuristics that lose more compression. Any time saved in detecting and eliminating conflicts is lost when writing a larger delta file out to storage.

2 Related Work

Encoding versions of data compactly by detecting altered regions of data is a well known problem. The first applications of delta compression found changed lines in text data for analyzing the recent modifications to files [11]. Considering data as lines of text fails to encode minimum sized delta files, as it does not examine data at a fine *granularity* and finds only matching data that are *aligned* at the beginning of a new line.

The problem of representing the changes between versions of data was formalized as string-to-string correction with block move [24] – detecting maximally matching regions of a file at an arbitrarily fine granularity without alignment. However, delta compression continued to rely on the alignment of data, as in database records [23], and the grouping of data into block or line granularity, as in source code control systems [22, 25], to simplify the combinatorial task of finding the common and different strings between versions.

Efforts to generalize delta compression to un-aligned data and to minimize the granularity of the smallest change resulted in algorithms for compressing data at the granularity of a byte. Early algorithms were based upon either dynamic programming [19] or the greedy method [24, 21, 17] and performed this task using time quadratic in the length of the input files.

Delta compression algorithms were improved to run in linear time and linear space. Algorithms with these properties have been derived from suffix trees [27, 18, 16] and as a generalization of Lempel-Ziv data compression [12, 13, 8]. Like algorithms based on greedy methods and dynamic programming, these algorithms generate optimally compact delta encodings.

Recent advances produced algorithms that run in linear time and constant space [3]. These differencing algorithms trade a small amount of compression, verified experimentally, in order to improve performance.

Any of the linear run-time algorithms allow delta compression to scale to large input files

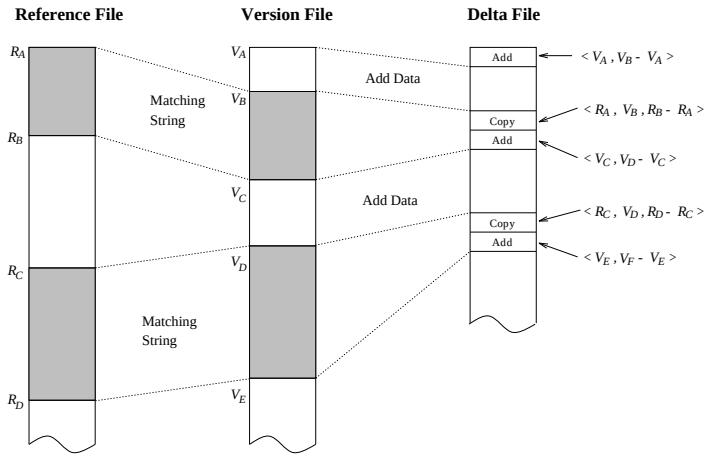


Figure 1: Encoding delta files. Common strings are encoded as *copy* commands $\langle f, t, l \rangle$ and new strings in the new file are encoded as *add* commands $\langle t, l \rangle$ followed by the string of length l of added data.

without known structure and permits the application of delta compression to file system backup and restore [6].

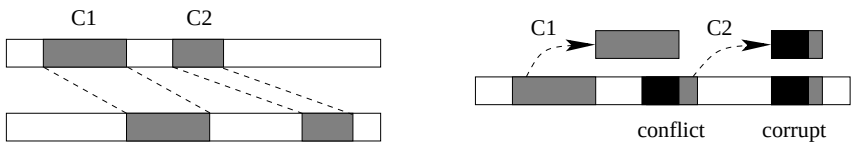
Recently, applications distributing HTTP objects using delta files have emerged [20, 4]. This permits web servers to both reduce the amount of data transmitted to a client and reduce the latency associated with loading web pages. Efforts to standardize delta files as part of the HTTP protocol and the trend toward making small network devices HTTP compliant indicate the need to distribute data to network devices efficiently.

3 Encoding Delta Files

Differencing algorithms encode the changes between two file versions compactly by finding strings common to both versions. We term these files a *version file* that contains the data to be encoded and a *reference file* to which the version file is compared. Differencing algorithms encode a file by partitioning the data in the version file into strings that are encoded using copies from the reference file and strings that are added explicitly to the version file (Figure 1). Having partitioned the version file, the algorithm outputs a delta file that encodes this version. This delta file consists of an ordered sequence of *copy* commands and *add* commands.

An *add* command is an ordered pair, $\langle t, l \rangle$, where t (to) encodes the string offset in the file version and l (length) encodes the length of the string. The l bytes of data to be added follow the command. A *copy* command is an ordered triple, $\langle f, t, l \rangle$ where f (from) encodes the offset in the reference file from which data are copied, t encodes the offset in the new file where the data are to be written, and l encodes the length of the data to be copied. The *copy* command moves the string data in the interval $[f, f + l - 1]$ in the reference file to the interval $[t, t + l - 1]$ in the version file.

In the presence of the reference file, a delta file rebuilds the version file with *add* and *copy* commands. The intervals in the version file encoded by these commands are disjoint. Therefore, any permutation of the command execution order materializes the same output version file.



(a) Delta copy (b) In-place copy

Figure 2: Data conflict and corruption when performing copy command C1 before C2.

4 In-Place Modification Algorithms

An in-place modification algorithm changes an existing delta file into a delta file that reconstructs correctly a new file version in the space the current version occupies. At a high level, our technique examines the input delta file to find *copy* commands that read from the write interval (file address range to which the command writes data) of other *copy* commands. The algorithm represents potential data conflicts in a digraph. The algorithm topologically sorts the digraph to produce an ordering on *copy* commands that reduces data conflicts. We eliminate the remaining conflicts by converting *copy* commands to *add* commands. The algorithm outputs the permuted and converted commands as an in-place reconstructible delta file. Actually, as described in more detail below, the algorithm performs permutation and conversion of commands concurrently.

4.1 Conflict Detection

Since we reconstruct files in-place, we concern ourselves with ordering commands that attempt to read a region to which another command writes. For this, we adopt the term *write before read* (*WR*) conflict [5]. For *copy* commands $\langle f_i, t_i, l_i \rangle$ and $\langle f_j, t_j, l_j \rangle$, with $i < j$, a *WR* conflict occurs when

$$[t_i, t_i + l_i - 1] \cap [f_j, f_j + l_j - 1] \neq \emptyset. \tag{1}$$

In other words, *copy* command i and j conflict if i writes to the interval from which j reads data. By denoting, for each *copy* command $\langle f_k, t_k, l_k \rangle$, the command's read interval as $\mathit{Read}_k = [f_k, f_k + l_k - 1]$ and its write interval as $\mathit{Write}_k = [t_k, t_k + l_k - 1]$, we write the condition (1) for a WR conflict as $\mathit{Write}_i \cap \mathit{Read}_j \neq \emptyset$. In Figure 2, commands C1 and C2 executed in that order generate a data conflict (blacked area) that corrupts data when a file is reconstructed in place.

This definition considers only *WR* conflicts between *copy* commands and neglects *add* commands. *Add* commands write data to the version file; they do not read data from the reference file. Consequently, an algorithm avoids all potential *WR* conflicts associated with adding data by placing *add* commands at the end of a delta file. In this way, the algorithms completes all reads associated with *copy* commands before executing the first *add* command.

Additionally, we define *WR* conflicts so that a *copy* command cannot conflict with itself. Yet, a single *copy* command's read and write intervals intersect sometimes and would seem to cause a conflict. We deal with read and write intervals that overlap by performing the copy in a *left-to-right* or *right-to-left* manner. For command $\langle f, t, l \rangle$, if $f \geq t$, we copy the string byte by byte starting at the left-hand side when reconstructing the original file. Since, the f (from) offset always exceeds the t (to) offset in the new file, a *left-to-right* copy never reads a byte over-written by a previous byte in the string. When $f < t$, a symmetric argument shows that we should start our copy at the

right hand edge of the string and work backwards. For this example, we performed the copies in a byte-wise fashion. However, the notion of a left-to-right or right-to-left copy applies to moving a read/write buffer of any size.

To avoid *WR* conflicts and achieve the in-place reconstruction of delta files, we employ the following three techniques.

1. Place all *add* commands at the end of the delta file to avoid data conflicts with *copy* commands.
2. Permute the order of application of the *copy* commands to reduce the number of write before read conflicts.
3. For remaining *WR* conflicts, remove the conflicting operation by converting a *copy* command to an *add* command and place it at the end of the delta file.

For many delta files, no possible permutation eliminates all *WR* conflicts. Consequently, we require the conversion of *copy* commands to *add* commands to create correct in-place reconstructible files for all inputs.

Having processed a delta file for in-place reconstruction, the modified delta file obeys the property

$$(\forall j)\left[\mathit{Read}_j \cap \left(\bigcup_{i=1}^{j-1} \mathit{Write}_i\right) = \emptyset\right], \tag{2}$$

indicating the absence of *WR* conflicts. Equivalently, it guarantees that a *copy* command reads and transfers data from the original file.

4.2 CRWI Digraphs

To find a permutation that reduces *WR* conflicts, we represent potential conflicts between the *copy* commands in a digraph and topologically sort this digraph. A topological sort on digraph $G = (V, E)$ produces a linear order on all vertices so that if G contains edge $\vec{uv}$ then vertex u precedes vertex v in topological order.

Our technique constructs a digraph so that each *copy* command in the delta file has a corresponding vertex in the digraph. On this set of vertices, we construct an edge relation with a directed edge $\vec{uv}$ from vertex u to vertex v when *copy* command u's read interval intersects *copy* command v's write interval. Edge $\vec{uv}$ indicates that by performing command u before command v, the delta file avoids a WR conflict. We call a digraph obtained from a delta file in this way a *conflicting read write interval* (*CRWI*) digraph. A topologically sorted version of this graph adheres to the requirement for in-place reconstruction (Equation 2).

4.3 Strategies for Breaking Cycles

As total topological orderings are possible only on acyclic digraphs and CRWI digraphs may contain cycles, we enhance a standard topological sort to break cycles and output a total topological order on a *subgraph*. Depth-first search implementations of topological sort [10] are modified easily to detect cycles. Upon detecting a cycle, our modified sort breaks the cycle by removing a vertex. When completing this enhanced sort, the sort outputs a digraph containing a subset of all

vertices in topological order and a set of vertices that were removed. This algorithm re-encodes the data contained in the *copy* commands of the removed vertices as *add* commands in the output.

As the string that contains the encoded data follows converted *add*, this replacement reduces compression in the delta file. We define the amount of compression lost upon deleting a vertex to be the *cost* of deletion. Based on this cost function, we formulate the optimization problem of finding the minimum cost set of vertices to delete to make a digraph acyclic. A *copy* command is an ordered triple $\langle f, t, l \rangle$. An *add* command is an ordered double $\langle t, l \rangle$ followed by the l bytes of data to be added to the new version of the file. Replacing a *copy* command with an *add* command increases the delta file size by $l - \|f\|$, where $\|f\|$ denotes the size of the encoding of offset f. Thus, the vertex that corresponds to the copy command $\langle f, t, l \rangle$ is assigned cost $l - \|f\|$.

When converting a digraph into an acyclic digraph by deleting vertices, an in-place conversion algorithm minimizes the amount of compression lost by selecting a set of vertices with the smallest total cost. This problem, called the FEEDBACK VERTEX SET problem, was shown by Karp [14] to be NP-hard for general digraphs. We have shown previously [7] that it remains NP-hard even when restricted to CRWI digraphs. Thus, we do not expect an efficient algorithm to minimize the cost in general.

For our implementation of in-place conversion, we examine two efficient, but not optimal, policies for breaking cycles. The *constant-time* policy picks the "easiest" vertex to remove, based on the execution order of the topological sort, and deletes this vertex. This policy performs no extra work when breaking cycles. The *local-minimum* policy detects a cycle and loops through all vertices in the cycle to determine and then delete the minimum cost vertex. The local-minimum policy may perform as much additional work as the total length of cycles found by the algorithm: $\mathrm{O}(n^2)$. Although these policies perform well in our experiments, we have shown previously [7] that they do not guarantee that the total cost of deletion is within a constant factor of the optimum.

4.4 Generating Conflict Free Permutations

Our algorithm for converting delta files into in-place reconstructible delta files takes the following steps to find and eliminate *WR* conflicts between a reference file and the new version to be materialized.

Algorithm

1. Given an input delta file, we partition the commands in the file into a set C of *copy* commands and a set A of *add* commands.
2. Sort the *copy* commands by increasing write offset, $C_{\text{sorted}} = \{c_1, c_2, ..., c_n\}$. For c_i and c_j, this set obeys: $i < j \longleftrightarrow t_i < t_j$. Sorting the copy commands allows us to perform binary search when looking for a copy command at a given write offset.
3. Construct a digraph from the *copy* commands. For the *copy* commands $c_1, c_2, ..., c_n$, we create a vertex set $V = \{v_1, v_2, ..., v_n\}$. Build the edge set E by adding an edge from vertex v_i to vertex v_j when *copy* command c_i reads from the interval to which c_j writes:

$$\overrightarrow{v_i v_j} \longleftrightarrow \textit{Read}_i \cap \textit{Write}_j \neq \emptyset \longleftrightarrow [f_i, f_i + l_i - 1] \cap [t_j, t_j + l_j - 1] \neq \emptyset.$$

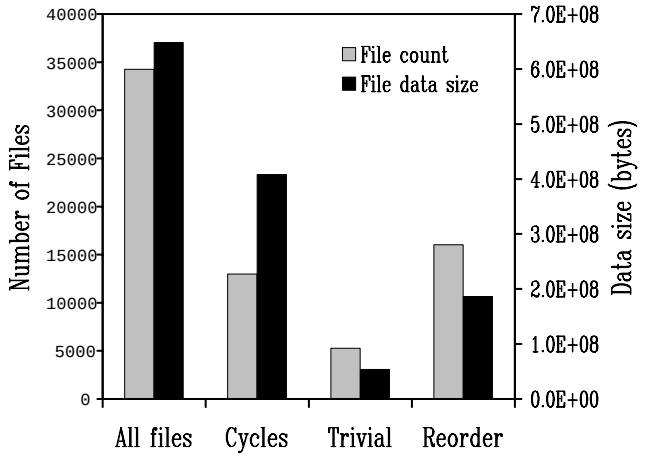


Figure 3: File counts and data size.

4. Perform a topological sort on the vertices of the digraph. This sort also detects cycles in the digraph and breaks them. When breaking a cycle, select one vertex on the cycle, using either the local-minimum or constant-time cycle breaking policy, and remove it. We replace the data encoded in its *copy* command with an equivalent *add* command, which is put into set A. The output of the topological sort orders the remaining *copy* commands so that they obey the property in Equation 2.
5. Output all *add* commands in the set A to the delta file.

The resulting delta file reconstructs the new version *out of order*, both out of write order in the version file and out of the order that the commands appeared in the original delta file.

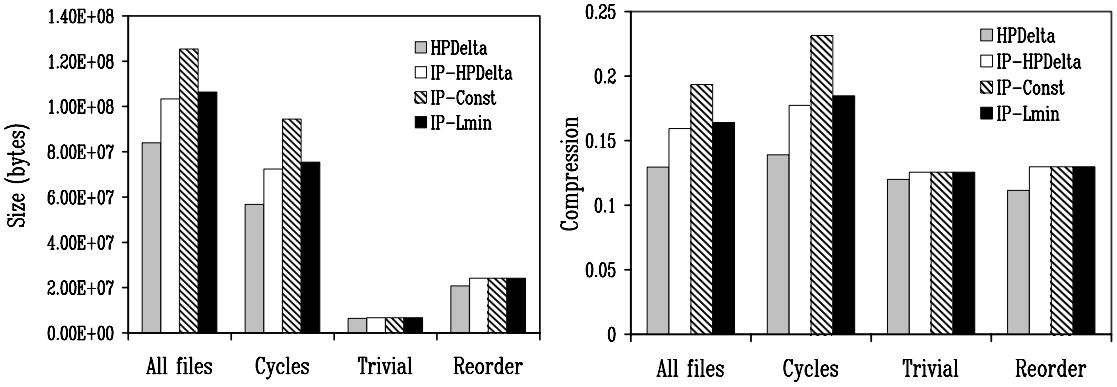
5 Experimental Results

As we are interested in using in-place reconstruction to distribute software, we extracted a large body of Internet available software and examined the compression and execution time performance of our algorithm on these files. Sample files include multiple versions of the GNU tools and the BSD operating system distributions, among other data, with both binary and source files being compressed and permuted for in-place reconstruction. These data were examined with the goals of:

- determining the compression loss due to making delta files in-place reconstructible;
- comparing the the constant-time and local-minimum policies for breaking cycles;
- showing in-place conversion algorithms to be efficient when compared with delta compression algorithms on the same data; and
- characterizing the graphs created by the algorithm.

In all cases, we obtained the original delta files using the correcting 1.5-pass delta compression algorithm [3].

We categorize the delta files in our experiments into 3 groups that describe what operations were require to make files in-place reconstructible. Experiments were conducted over more than



(a) Delta size

(b) Delta compression

Figure 4: Compression performance

34,000 delta files totaling 6.5MB (Megabytes). Of these files (Figure 3), 63% of the files contained cycles that needed to be broken. 29% did not have cycles, but needed to have *copy* commands reordered. The remaining 8% of files were trivially in-place reconstructible; *i.e.*, none of the *copy* commands conflicted. For trivial files, performing copies before adds creates an in-place delta.

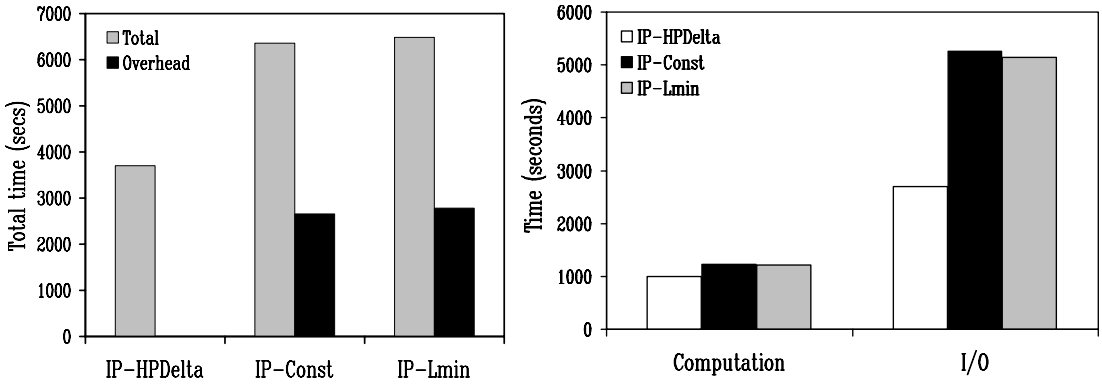
The amount of data in files is distributed differently across the three categories than are the file counts. Files with cycles contain over 4MB of data with an average file size of 31.4KB. Files that need copy commands reordered hold 1.9MB of data, with an average file size of 11.6KB. Trivially in-place reconstructible files occupy 585KB of data with an average file size of 10.2KB.

The distribution of files and data across the three categories confirms that efficient algorithms for cycle breaking and command reordering are needed to deliver delta compressed data in-place. While most delta files do not contain cycles, those that do have cycles contain the majority of the data.

We group compression results into the same categories. Figure 4(a) shows the relative size of the delta files and Figure 4(b) shows compression (size of delta files as a fraction of the original file size). For each category and for all files, we report data for four algorithms: the unmodified correcting 1.5-pass delta compression algorithm [3] (HPDelta); the correcting 1.5-pass delta compression algorithm modified so that code-words are in-place reconstructible (IP-HPDelta); the in-place modification algorithm using the local-minimum cycle breaking policy (IP-Lmin); and the in-place modification algorithm using the constant-time cycle breaking policy (IP-Const).

The HPDelta algorithm is a linear time, constant space algorithm for generating delta compressed files. It outputs *copy* and *add* commands using a code-word format similar to industry standards [15].

The IP-HPDelta algorithm is a modification of HPDelta to output code-words that are suitable for in-place reconstruction. Throughout this paper, we have described *add* commands $\langle t, l \rangle$ and *copy* commands $\langle f, t, l \rangle$, where both commands encode explicitly the to t or write offset in the version file. However, delta algorithms that reconstruct data in write order need not explicitly encode a write offset – an *add* command can simply be $\langle l \rangle$ and a *copy* command $\langle f, l \rangle$. Since commands are applied in write order, the end offset of the previous command implies the write offset of the current command implicitly. The code-words of IP-HPDelta are modified to make the write offset explicit. The explicit write offset allows our algorithm to reorder copy commands. This extra field in each code-word introduces a per-command overhead in a delta file. The amount



(a) Overhead

(b) Breakdown

Figure 5: Run-time results

of overhead varies, depending upon the number of commands and the original size of the delta file. Encoding overhead incurs a 3% compression loss over all files.

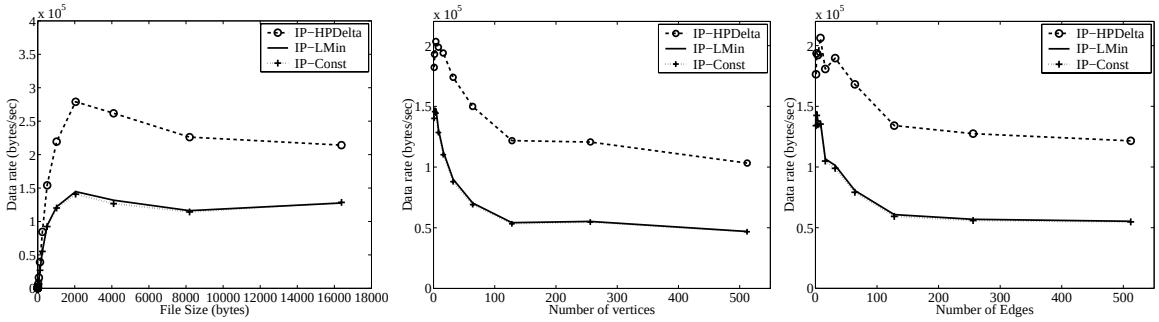
From the IP-HPDelta algorithm, we derive the IP-Const and IP-Lmin algorithms. They run the IP-HPDelta algorithm to generate a delta file and then permute and modify the commands according to our technique to make the delta file in-place reconstructible. The IP-Const algorithm implements the constant-time policy and the IP-Lmin algorithm implements the local-minimum policy.

Experimental results indicate the amount of compression lost due to in-place reconstruction and divides the loss into encoding overhead and cycle breaking. Over all files, HPDelta compresses data to 12.9% its original size. IP-HPDelta compresses data to 15.9%, losing 3% compression to encoding overhead. IP-Const loses an additional 3.4% compression by breaking cycles for a total compression loss of 6.4%. In contrast, IP-Lmin loses less than 0.5% compression for a total loss of less than 3.5%. The local-minimum cycle breaking policy performs excellently in practice, because compression losses are small when compared with encoding overheads. With IP-Lmin, cycle breaking accounts for less than 15% of the loss. IP-Const more than doubles the compression loss.

For reorder and trivial in-place delta files, no cycles are present and no compression lost. Encoding overhead makes up all lost compression – 0.5% for trivial delta files and 1.8% for reordered files.

Files with cycles exhibit an encoding overhead of 3.8% and lose 5.4% and 0.7% to cycle breaking for the IP-Const and IP-Lmin respectively. Because files with cycles contain the majority of the data, the results for files with cycles dominate the results for all files.

In-place algorithms incur execution time overheads when performing additional I/O and when permuting the commands in a delta file. An in-place algorithm must generate a delta file and then modify the file to have the in-place property. Since a delta file does not necessarily fit in memory, in-place algorithms create an intermediate file that contains the output of the delta compression algorithm. This intermediate output serves as the input for the algorithm that modifies/permutes commands. We present execution-time results in Figure 5(a) for both in-place algorithms – IP-Const and IP-Lmin. IP-Lmin and IP-Const perform all of the steps of the base algorithm (IP-HPDelta) before manipulating the intermediate file. Results show that the extra work incurs an



(a) File Size (b) Vertices (c) Edges

Figure 6: Run-time results

overhead of about 75%. However, figure 5(b) shows that almost all of this overhead comes from additional I/O. We conclude that the algorithmic tasks for in-place reconstruction are small when compared with the effort compressing data (about 10% the run-time) and miniscule compared to the costs of performing file I/O.

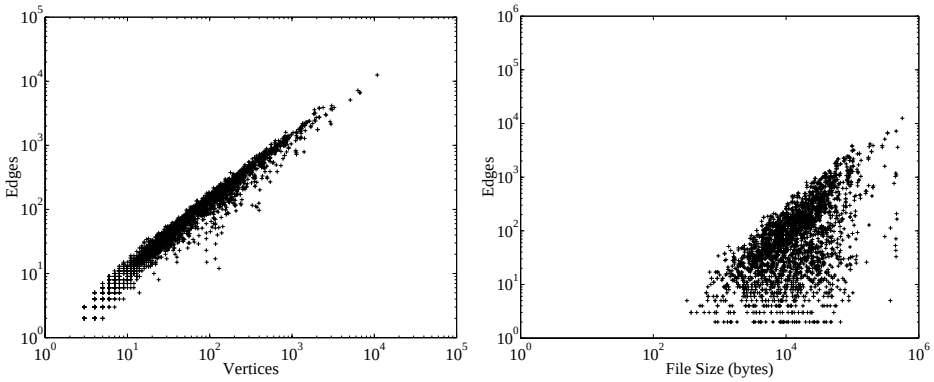
Despite inferior worst-case run-time bounds, the local-minimum cycle breaking policy runs faster than the constant-time policy in practice. Because file I/O dominates the run-time costs and because IP-Lmin creates a smaller delta file, it takes less total time than the theoretically superior IP-Const. In fact, IP-Const spends 2.2% more time performing I/O as a direct result of the files being 2.9% larger. IP-Lmin even uses slightly less time performing computation than IP-Const, which has to manipulate more data in memory.

Examining run-time results in more detail continues to show that IP-Lmin outperforms IP-Const, even for the largest and most complex input files. In Figure 6, we see how run-time performance varies with the input file size and with the size of the graph the algorithm creates (number of edges and vertices); these plots measure run time by data rate – file size (bytes) divided by run time (seconds).

Owing to start-up costs, data rates increase with file size up to a point, past which rates tend to stabilize. The algorithms must load and initialize data structures. For small files, these costs dominate, and data rates are lower and increase linearly with the file size (Figure 6(a)). For files larger than 2000 bytes, rates tend to stabilize, exhibiting some variance, but neither increasing or decreasing as a trend. These results indicate that for inputs that amortize start-up costs, in-place algorithms exhibit a data rate that does not vary with the size of the input – a known property of the HPDelta algorithm [3]. IP-Lmin performs slightly better than IP-Const always.

The performance of all algorithms degrades as the size of the CRWI graphs increase. Figure 6(b) shows the relative performance of the algorithms as a function of the number of vertices, and Figure 6(c) shows this for the number of edges. For smaller graphs, performance degrades quickly as the graph size increases. For larger graphs, performance degrades more slowly. The graph size corresponds directly to the number of copy commands in a delta file. The more commands, the more I/O operations the algorithm must execute. Often more vertices means more small I/O rather than fewer large I/O, resulting in lower data rates.

Surprisingly, IP-Lmin continues to out-perform IP-Const even for the largest graphs. Analysis would indicate that the performance of IP-Lmin and IP-Const should diverge as the number of



(a) Edges versus Vertices (b) Edges versus File Size

Figure 7: Edges in delta files that contain cycles.

edges increase. But no evidence of divergent performance exists. We attribute this to two factors: (1) graphs are relatively small and (2) all algorithms are I/O bound.

In Figure 7, we look at some statistical measures of graphs constructed when creating in-place delta files, restricted to those graphs that contain cycles. While graphs can be quite large, a maximum of 11503 vertices and 16694 edges, the number of edges scales linearly with the number of vertices and less than linearly with input file size. The constructed graphs do not exhibit edge relations that approach the $O(|V|^2)$ upper bound. Therefore, data rate performance should not degrade as the number of edges increases. For example consider two files as inputs to the IP-Lmin algorithm – one with a graph that contains twice the edges of the other. Based on our result, we expect the larger graph to have twice as many vertices and encode twice as much data. While the larger instance does twice the work breaking cycles, it benefits from reorganizing twice as much data, realizing the same data rate.

The linear scaling of edges with vertices and file size matches our intuition about the nature of delta compressed data. Delta compression encodes multiple versions of the same data. Therefore, we expect matching regions between these files (encoded as edges in a CRWI graph) to have spatial locality; *i.e.*, the same string often appears in the same portion of a file. These input data do not exhibit correlation between all regions of a file which would result in dense edge relations. Additionally, delta compression algorithms localize matching between files, correlating or synchronizing regions of file data [3]. All of these factors result in the linear scaling that we observe.

6 Conclusions

We have presented algorithms that modify delta files so that the encoded version may be reconstructed in the absence of scratch memory or storage space. Such an algorithm facilitates the distribution of software to network attached devices over low bandwidth channels. Delta compression lessens the time required to transmit files over a network by encoding the data to be transmitted compactly. In-place reconstruction exchanges a small amount of compression in order to do so without scratch space.

Experimental results indicate that converting a delta file into an in-place reconstructible delta file has limited impact on compression, less than 4% in total with the majority of compression

loss from encoding overheads rather than modifications to the delta file. We also find that for bottom line performance keeping delta files small to reduce I/O matters more than execution time differences in cycles breaking heuristics, because in-place reconstruction is I/O bound. For overall performance, the algorithm to convert a delta file to an in-place reconstructible delta file requires less time than generating the delta file in the first place.

In-place reconstructible delta file compression provides the benefits of delta compression for data distribution to an important class of applications – devices with limited storage and memory. In the current network computing environment, this technology decreases greatly the time to distribute content without increasing the development cost or complexity of the receiving devices. Delta compression provides Internet-scale file sharing with improved version management and update propagation, and in-place reconstruction delivers the technology to the resource constrained computers that need it most.

7 Future Directions

Detecting and breaking conflicts at a finer granularity can reduce lost compression when breaking cycles. In our current algorithms, we eliminate cycles by converting *copy* commands into *add* commands. However, typically only a portion of the offending *copy* command actually conflicts with another command; only the overlapping range of bytes. We propose, as a simple extension, to break a cycle by converting part of a *copy* command to an *add* command, eliminating the graph edge (rather than a whole vertex as we do today), and leaving the remaining portion of the *copy* command (and its vertex) in the graph. This extension does not fundamentally change any of our algorithms, only the cost function for cycle breaking.

As a more radical departure from our current model, we are exploring reconstructing delta files with bounded scratch space, as opposed to zero scratch space as with in-place reconstruction. This formulation, suggested by Martín Abadi, allows an algorithm to avoid *WR* conflicts by moving regions of the reference file into a fixed size buffer, which preserves reference file data after that region has been written. The technique avoids compression loss by resolving data conflicts without eliminating *copy* commands.

Reconstruction in bounded space is logical, as target devices often have a small amount of available space that can be used advantageously. However, in-place reconstruction is more generally applicable. For bounded space reconstruction, the target device must contain enough space to rebuild the file. Equivalently, an algorithm constructs a delta file for a specific space bound. Systems benefit from using the same delta file to update software on many devices. For example, distributing an updated product list to many PDAs in the same sales force. In such cases, in-place reconstruction offers a lowest common denominator solution in exchange for a little lost compression.

We also are developing algorithms that can perform peer-to-peer style delta compression [26] in an in-place fashion. This allows delta compression to be used between two versions of a file stored on separate machines and is often a more natural formulation, because it does not require a computer to maintain the original version of data to employ delta compression. This works well for file systems, most of which do not handle multiple versions.

Our ultimate goal is to use in-place algorithms as a basis for a data distribution system. The system will operate both in hierarchical (client/server) and peer-to-peer modes. It will also conform

to Internet standards [15] and, therefore, work seamlessly with future versions of HTTP.

References

[1] The free network project – rewiring the Internet. Technical Report http://freenet.sourceforge.net/, 2001.

[2] The gnutella protocol specification. Technical Report http://www.gnutelladev.com/protocol/gnutella-protocol.html, 2001.

[3] M. Ajtai, R. Burns, R. Fagin, D. D. E. Long, and L. Stockmeyer. Compactly encoding unstructured input with differential compression. `www.almaden.ibm.com/cs/people/stock/diff7.ps`, IBM Research Report RJ 10187, April 2000 (revised Aug. 2001).

[4] G. Banga, F. Douglis, and M. Rabinovich. Optimistic deltas for WWW latency reduction. In *Proceedings of the 1998 Usenix Technical Conference*, 1998.

[5] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison–Wesley Publishing Co., 1987.

[6] R. C. Burns and D. D. E. Long. Efficient distributed backup and restore with delta compression. In *Proceedings of the Fifth Workship on I/O in Parallel and Distributed Systems, San Jose, CA*, November 1997.

[7] R. C. Burns and D. D. E. Long. In-place reconstruction of delta compressed files. In *Proceedings of the Seventeenth ACM Symposium on Principles of Distributed Computing*, 1998.

[8] M. Chan and T. Woo. Cache-based compaction: A new technique for optimizing web transfer. In *Proceedings of the IEEE Infocom '99 Conference, New York, NY*, March 1999.

[9] S. S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, May 1997.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[11] S. P. de Jong. Combining of changes to a source file. *IBM Technical Disclosure Bulletin*, 15(4):1186–1188, September 1972.

[12] J. J. Hunt, K.-P. Vo, and W. F. Tichy. An empirical study of delta algorithms. In *Proceedings of the 6th Workshop on Software Configuration Management*, March 1996.

[13] J. J. Hunt, K.-P. Vo, and W. F. Tichy. Delta algorithms: An empirical analysis. *ACM Transactions on Software Engineering and Methodology*, 7(2):192–214, 1998.

[14] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104. Plenum Press, 1972.

[15] D. G. Korn and K.-P. Vo. The VCDIFF generic differencing and compression format. Technical Report Internet-Draft draft-vo-vcdiff-00, Internet Engineering Task Force (IETF), 1999.

[16] S. Kurtz. Reducing the space requirements of suffix trees. *Software – Practice and Experience*, 29(13):1149–1171, 1999.

[17] J. P. MacDonald, P. N. Hilfinger, and L. Semenzato. PRCS: The project revision control system. In *Proceedings System Configuration Management*, 1998.

[18] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2), April 1978.

[19] W. Miller and E. W. Myers. A file comparison program. *Software – Practice and Experience*, 15(11):1025–1040, November 1985.

[20] J. C. Mogul, F. Douglis, A. Feldman, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *Proceedings of ACM SIGCOMM '97*, September 1997.

[21] C. Reichenberger. Delta storage for arbitrary non-text files. In *Proceedings of the 3rd International Workshop on Software Configuration Management, Trondheim, Norway, 12-14 June 1991*, pages 144–152. ACM, June 1991.

[22] M. J. Rochkind. The source code control system. *IEEE Transactions on Software Engineering*, SE-1(4):364–370, December 1975.

[23] D. G. Severance and G. M. Lohman. Differential files: Their application to the maintenance of large databases. *ACM Transactions on Database Systems*, 1(2):256–267, September 1976.

[24] W. F. Tichy. The string-to-string correction problem with block move. *ACM Transactions on Computer Systems*, 2(4), November 1984.

[25] W. F. Tichy. RCS – A system for version control. *Software – Practice and Experience*, 15(7):637–654, July 1985.

[26] A. Tridgell and P. Mackerras. The RSync algorithm. Technical Report TR-CS-96-05, The Australian National University, 1996.

[27] P. Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

Intra-file Security for a Distributed File System

Scott A. Banachowski, Zachary N. J. Peterson, Ethan L. Miller and Scott A. Brandt
Storage Systems Research Center
Jack Baskin School of Engineering
University of California
Santa Cruz, CA 95064
{sbanacho,zachary,elm,scott}@cs.ucsc.edu
Telephone: +1 (831) 459-2545
Fax: +1 (831) 459-4829

Abstract

Cryptographic file systems typically provide security by encrypting entire files or directories. This has the advantage of simplicity, but does not allow for fine-grained protection of data within very large files. This is not an issue in most general-purpose systems, but can be very important in scientific applications where some but not all of the output data is sensitive or classified. We present a more flexible approach that uses common cryptographic techniques to secure any arbitrary-sized region of data within a file, even if the region is logically non-contiguous. This approach, called intra-file encryption, allows mixing data of different sensitivity in a single file. This benefits users by permitting related data belonging to a single file to be kept together rather than separating data of different security needs. Supporting intra-file encryption requires additional file metadata and key management services. For file systems that store metadata and files on the same server, the management of extra metadata poses little problem beyond storage overhead. However, for high-performance network-attached file systems, the additional metadata poses greater challenges related to data placement and security. This paper describes the intra-file security encryption technique with discussion of including support for it in a distributed file system.

1 Introduction

Traditionally, file system security uses an "all-or-nothing" approach—all of a file is encrypted identically. This approach is sufficient in situations where a file must be accessed in its entirety to make sense for a user or application. However, there are many cases where a user should only have access to some of the data in a file. A large file used for scientific modeling might contain mostly unclassified information, with some sections of classified

data. Other examples include a satellite map of a region containing military zones, a specification for a vehicle with sensitive information, or a recipe with a secret ingredient. Using current techniques, users that desire different levels of security must use different files, complicating access for all users.

In this paper, we introduce *intra-file security*—a flexible approach to providing end-to-end encryption in a file system. It allows users to encrypt extents of files independently from other extents, so that a single file may contain one or more secure regions. A file system incorporating intra-file security transparently handles most operations, such as automatic decryption and key management. The result is a file system with little extra programming or runtime overhead for the added functionality. Reads are entirely managed by the file system and writes occur via two separate but nearly identical function calls for unencrypted and one for encrypted data.

Flexible end-to-end encryption technology is becoming increasingly important as systems use distributed storage architectures. High-performance computer systems deal with data sets of tremendous size; files used in scientific computing and data-mining applications commonly extend beyond the capabilities of single storage devices. Distributed storage architectures provide one solution for the demands of increased storage needs. By spreading file system data over multiple network nodes, distributed storage provides high data rates through parallelism, and large, scalable storage capacity with a capability for fault tolerance through redundancy. However, distributing storage also increases the number of potential points for network intrusion, making data susceptible to security breaches. To secure sensitive data, networked file servers should store and transmit only encrypted data, which is decoded by clients with cryptographic keys. Many end-to-end encryption tools exist, and the least cumbersome for users are those built into the file system [1]. Such file systems transparently decode encrypted data for users with proper permission rights.

Existing cryptographic file systems secure data on a per-directory [1] or per-file [4] basis. This level of granularity is not flexible enough to support applications that benefit from encrypting smaller regions within files. If information is only encrypted on a per-file basis, then a set of data containing a mix of sensitive and unclassified data must be stored in two or more files, one for each security level. However, in some cases it is beneficial to keep data in a single file; users and tools can manage the data as a single entity in the file system, and the same applications may use secure and insecure data sets. Because they encrypt whole files or file systems, existing cryptographic file system techniques cannot address this problem.

Intra-file security offers additional security by allowing more fine-grained control file access, breaking a file into regions of differing security without compromising single-file semantics. This allows the system to transparently handle security operations, making the security invisible to authorized users and thus more likely to actually be used. In order to implement intra-file security, we introduce security-related metadata, and provide a key management solution that allows flexibility in security and access policy.

Section 2 introduces the intra-file security (IFS) encryption algorithm. The algorithm, based on well-known cryptographic techniques, may be implemented stand-alone or as

part of a larger system, such as a file system. Section 3 describes how to integrate IFS into a distributed object-based file system. Sections 4 and 5 discuss some possible IFS applications and related work.

2 Intra-File Security

Intra-file security (IFS) allows encryption to be applied to segments as small as a byte or as large as an entire file; multiple encrypted segments need not be logically contiguous within the file. In an IFS file, encrypted data is stored logically in-place, and occupies the physical file blocks that would have contained the unencrypted data. To support efficient random file access, we independently encrypt data from each logical file block, so there is no dependence on information from other blocks. Consider the file shown in Figure 1, which contains a non-contiguous region that must be kept secure. The region spans one entire logical block (L_1), and two partial blocks (L_2 and L_3). As mentioned above, this region is not independently encryptable using standard techniques. With IFS, this non-contiguous region of the file can be encrypted independently and made available only to appropriate users. Furthermore, because the encrypted data is left in place, all programs written to work with the full data set (including legacy applications) can still function properly. All regions of the data, encrypted and unencrypted alike, will still be readable except that the encrypted regions will not contain the secured data values but will instead contain apparently random values.

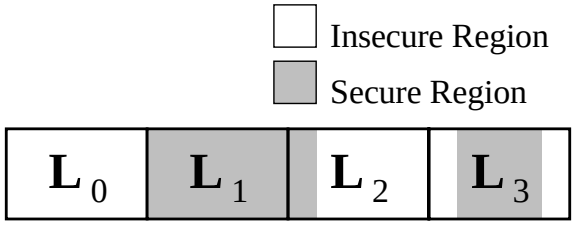


Figure 1: A single logical file address space broken into secure and insecure regions.

The encryption technique may use any block or stream cryptographic algorithm. Because the size of encrypted data in a file block may not match cipher block sizes, the algorithm is well-suited to stream ciphers, but can also be made to work with block ciphers with little additional effort. The flexibility of choosing any cryptographic algorithm allows system builders to vary encryption strength, conform with specific standards, or integrate off-the-shelf hardware chips into the system. The choice of block or stream cipher presents only a slight variation on the technique, so we present methods for both.

2.1 Block Cipher Technique

In an IFS file, secure segments may reside anywhere within a block, and may not be physically contiguous within a block. This causes a problem for block encryption algorithms

that expect to receive contiguous blocks of data for encryption. Our system combines all segments within a block into a temporary buffer before encryption, encrypts the buffer, and then redistributes the cipher back into the positions of the original plain-text segments. This process uses scatter-gather, minimizing actual copies to the bytes at the start and end of a region necessary to pad out the encryption block (often 64–128 bits), and uses pointer manipulation to do the rest of the encryption in place.

Because the output of a block algorithm is a fixed size, and the data may not necessarily match this size, we employ *cipher-text stealing* [2] to match encrypted data sizes to unencrypted sizes. Cipher-text stealing allows us to output ciphers of the same size as the input, even if they do not match the cipher block size. The encrypted data is then redistributed back to the file block in the area originally occupied by its plain-text counterpart. By using *initialization vectors* (IVs) [13] and *cipher block chaining* (CBC) [13], we also obscure data containing repeated patterns (such as headers) The IV must be unique for each block in a storage device but need not be secret.

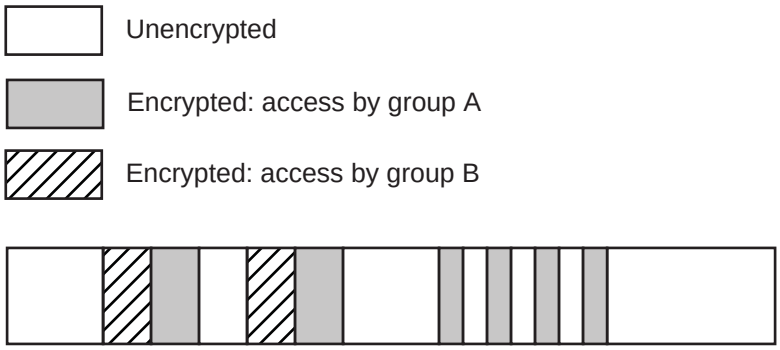
2.2 Stream Cipher Technique

By using a stream cipher such as RC4 or SEAL [13], IFS does not need to assemble data into temporary buffers or use pointer manipulation to collect bytes for encryption; instead, data may be encrypted in place. Stream ciphers such as RC4 claim a speed improvement of 10 times over DES, further improving performance. Applying feedback chaining to the stream hides data patterns—we use an IV to initialize the feedback chain, therefore the metadata structure does not differ from block mode encryption.

2.3 Encryption Metadata

By default, all data in the file is assumed to be unencrypted. In order to locate the secure data within the file, and to find the encryption parameters, each encrypted block requires a description of the location of secure segments and initialization vector information. In IFS, the structure holding this data is a security node, or *s-node*, shown in Figure 2. The size of an s-node depends on the number and layout of secure regions. A secure region is defined by an extent consisting of a start and a length; the start is relative to the start of the previous secure region, or the start of the block for the first region. Because many secure regions are formed of repeating patterns of data of varying levels of security, there is also a shorthand way of representing simple patterns of secure regions that are a fixed length and fixed distance apart. This is accomplished by specifying a repetition count associated with the offset and length specified in the secure region specification.

In addition to information about the location of secure regions, s-nodes must store the information necessary to encrypt and decrypt the secured data. This includes key information for the region as well as an initialization vector (IV)—a number used to seed the encryption algorithm when it operates on the encrypted data in the block. An IV is necessary to ensure that encrypted regions with the same data do not result in the same ciphertext, providing



Start	Length	Count	s-group
512	256	1	A
256	256	1	B
512	256	1	A
256	256	1	B
768	128	1	A
256	128	3	A

Figure 2: A 4 KB block encrypted with intra-file security and its associated security node (s-node). Note that the last entry in the s-node has a repeat count of 3, representing the three repeated secure regions near the end of the file. The first of the four regions must be represented separately because its distance from the previous region is larger than that of the following three regions.

insight about the file's structure or contents that might prove useful to an intruder. The IV must differ for each file block, and thus is a function of the logical block number as well as per-file values such as file identifier. If the IV for a block can be determined *solely* from the logical block number and per-file constants, it need not be stored in the s-node because it can be calculated at runtime.

Pointers to keys, on the other hand, must always be stored in the s-nodes. It might be possible to avoid storing key information in the s-node by simply referring to key information for the whole file; however, this approach would not permit encrypting portions of a file with different keys. Instead, we store an *s-group* identifier for each secure region; this identifier is translated by the system into a key using the approach discussed in Section 3.1.

There is one s-node structure for each logical file block that contains any encrypted segments. Note, however, that it is possible to group file system blocks together to reduce the amount of storage required by s-nodes; this technique is particularly effective for files that require large numbers of identically-sized regions with constant spacing. In such files, a few secure region descriptors can suffice for a large number of secure regions, reducing the file system overhead for IFS. Because s-nodes are allocated by the file system from the same pool of blocks used for regular files, reducing the size of security information allows more data to be stored in the file system.

It should be noted that while they are adequate for their intended purpose, the s-node structure described in this section could be improved in several ways. The s-node as depicted in Figure 2 is simple to implement, but uses space inefficiently. Instead, s-nodes could be compressed using gamma compression [14] or other techniques for compressing small numbers. Additionally, an IFS system could attempt to recognize and represent more complex encryption patterns, albeit at the cost of added complexity.

3 Integration with an OBSD File System

Although IFS may be used in any type of file system, we present a design to implement intra-file security for a file system based on Object Based Storage Devices (OBSDs). We are proposing the use of OBSDs for high-performance network-attached storage devices; this approach has similarities to Network-Attached Secure Disks (NASD) [3]. An OBSD-based file system is designed for high-performance computing workloads—precisely the kinds of applications that benefit from intra-file security. Because OBSDs require strong security in order to keep data safe in storage and transit [7], we expand the end-to-end encryption capabilities by incorporating IFS.

OBSD-based storage systems have the potential to improve both file system performance and functionality by building a high-performance storage system from inexpensive storage components connected by high-speed networks. The main hardware component of the storage system is an object-based storage device—one or more disks (or other storage devices) managed by a single CPU and seen by the file system as a single device. Data is distributed across many OBSDs, with high bandwidth coming from large numbers of concurrently operating OBSDs.

Each OBSD is responsible for managing and allocating its own storage; requests to an OBSD are of the form "write (or read) this range of bytes from file X," with low-level placement of the data and free space management left to the OBSD. High-level information such as the striping pattern across OBSDs and translation of names to file identifiers are left to a metadata server (MS), which is accessed by the user only when a file is opened or closed. This file system design is shown in Figure 3.

The key advantage of OBSDs in a high-performance environment is the ability to delegate low-level block allocation and synchronization for a given segment of data to the device on which it is stored, leaving the file system to decide only on which OBSD a particular segment should be placed. In such a distributed file system, s-nodes are stored physically near the blocks they describe, avoiding extra traffic to central servers on distributed storage systems and amortizing I/O usage among the devices. OBSDs use their own allocation policies to manage local data, including file and s-node data, placing them for efficiency within physical storage devices. Because s-nodes do not contain secrets, end-to-end encryption is provided to users without any extra involvement of the OBSD—the OBSD sends all file data and s-nodes in the clear on insecure networks. The security of encrypted data lies with the key management policy.

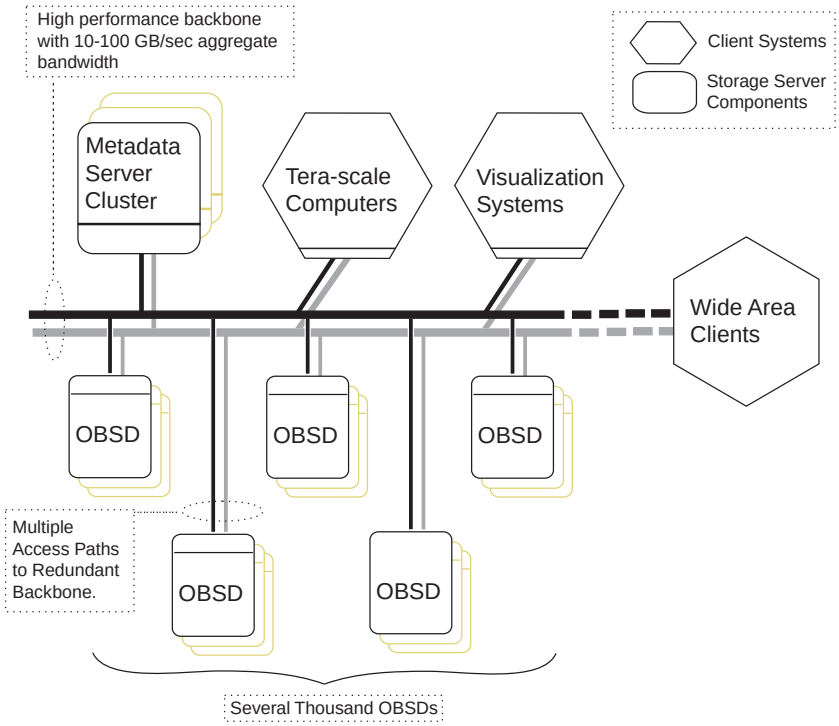


Figure 3: OBSD storage system architecture.

3.1 Authentication and Key Management

An authentication system is required for file system security, regardless of end-to-end encryption capabilities. Since we are focusing on support for intra-file encryption, a full development of the authentication system is beyond the scope of this paper. However, we rely on an authentication system for distribution of encryption keys, so we briefly describe how such a system may be implemented.

A major role of a metadata server (MS) is to control access to the file system. When users wish to open a file, the MS checks file permissions before granting access. As a first step, client software authenticates a user's identity, using standard authentication techniques such as Kerberos [9] or cryptographic hashes [7, 10]. The MS proceeds to check permission for a requested file operation using the file system's access control mechanism. However, OBSDs handle read and write requests directly; in order to enforce access rights, OBSDs must also check identities and permissions as well. The overhead of maintaining and checking access permissions at each OBSD defeats the high-performance requirement, so an OBSD uses a more efficient method to check the validity of a client's request. The MS generates *tokens* containing encoded access rights during open requests, and sends them to clients along with the file's metadata. Clients present these tokens with their requests to OBSDs. By checking the permissions encoded in the token, an OBSD determines the validity of the request. Tokens are equivalent to *capabilities* used in NASD for the same purpose [4, 3]. In IFS, security information is included in the forwarded tokens.

Access to encrypted segments is based on IFS group permissions, which we call *s-groups*. An s-group contains a list of users and/or groups that may use the key for an encrypted segment; the creator of an encrypted segment specifies s-group members during the initial write. A key server (KS) manages s-groups separately from file-access group permissions normally associated with file services; the goal is to remove management of encryption from traditional file system administration. The KS is responsible for checking s-group permissions, and generating and storing keys. From a user's viewpoint, calls to the MS involve both the MS and the KS, whether they reside on single or concurrent machines.

3.2 File I/O Interface

IFS uses standard POSIX file semantics by instrumenting interface libraries to handle security operations transparently. However, supporting encryption requires some new functions that allow writing of encrypted segments. Applications writing *only* unencrypted data and reading *any* data use the normal write and read function interfaces.

Reading encrypted data is transparent to the user. When reading data, users with a key see decrypted data when they read data; thus, applications reading data stored with IFS do not need any modifications, though they must be capable of dealing with garbage data in the data file—reads from encrypted segments of a file appear as random bits if the user lacks the proper key. If the user has the necessary key, the file system client transparently decrypts the file using keys supplied with authentication tokens. Only users with the proper key may decrypt secure segments and view the contents; the encoding of the token tells the OBSD whether or not to send s-nodes with data, so extra traffic is avoided when possible.

Under IFS, the interface to the file system is extended to support encrypted writes. Encrypted segments remain read-only unless the user has *encrypted write access*, which is granted through IFS s-group permissions. Even for users with permission, encrypted writes are explicit. Two new system calls support encrypted writes. One function translates an s-group specification into an integer identifier. The identifier is used in subsequent calls to the `secure_write` function, which is identical to the standard 'C' `write` function except for this additional argument. When writing encrypted segments, the file system client creates s-nodes for the corresponding blocks, and sends the s-nodes to the OBSD along with the blocks. When over-writing data in blocks already allocated to the file, the client must fetch and update the existing s-node (read-modify-write operation).

Unencrypted write requests to file blocks containing encryption must be carefully controlled, because users without encryption rights cannot overwrite the encrypted region of the block. To protect the integrity of encrypted data, it is impossible for users to write to encrypted segments using the traditional write function call. In order to minimize the latency of unencrypted writes, the OBSD quickly caches all data on writes, and during periods of inactivity discards changes to encrypted segments before committing the write. Essentially, this makes all encrypted segments read-only unless invoking the `secure_write` function. This policy does not impact blocks without encrypted segments, but it effects the coherency of blocks that do—until the write is fully committed, multiple copies of a block

reside in the file system. As a trade-off between performance and safety, we prefer that writes to encrypted segments do not occur unless made explicit, even for users with a key.

4 IFS Applications

To support encryption of data within existing unencrypted files that have been migrated to an IFS file system or written with non-IFS legacy applications, an IFS-capable copy program can be provided to encrypt the appropriate portions of the file. This program would take as input an unencrypted file and a specification of the regions to be encrypted.

Databases that use a single large flat-file could easily benefit from IFS by encrypting those fields of the database that must be kept secret, while still maintaining single-file semantics for the whole database. Most databases support encrypted fields by simply supplying keys for particular fields; however, this approach requires a reasonable amount of support from the database system or the database queries to remain transparent to users. By using IFS, this process could be made transparent, particularly if databases exchanged information with the file system.

Many very large files used in military and government scientific work will also benefit from IFS. Removing the need to fragment files that naturally require multiple levels of security will simplify applications as well as data management; no longer will users need to create several files with different encryption levels and keep track of which ones are related and how. Eliminating fragmentation ensures high-performance sequential and random access. Importantly, legacy applications can transparently be made IFS-capable, since the data formats and locations within the files remain unchanged even as portions of the data itself are encrypted.

IFS may also be used to transfer partial files in a distributed file system, as suggested by Muthitacharoen *et al.* [8]. By integrating IFS into a low-bandwidth distributed file system, users could gain secure access to their files even from slow clients.

5 Related Work

There have been many file systems and storage systems that provide higher security by encrypting files and metadata. Reidel, *et al.* [11] provide a good framework for evaluating secure file systems; their work discusses file systems and the security that each provides. Intra-file security is not one of their criteria; although they do discuss the granularity of key protection, the minimum protection unit is a single file.

Some file systems, such as CFS [1] and Cryptfs [15], require users to manage their own keys. This approach is simple, but is not suitable for IFS because of the sheer number of keys required [12]. Other systems such as SNAD [7], SFS and SUNDR [6, 5], and NASD [3] automatically manage encryption keys, though they do not permit partial-file encryption. Moreover, many of these systems, including NASD and SFS, store data on

the disk in an unencrypted form, using encryption only for authentication. The techniques described in this paper are based on those used in SNAD—it provides strong protection by encrypting data end-to-end, leaving it in the clear only on the client.

Intra-file security is particularly important for large, distributed file systems such as those enabled by NASD [3] and object-based storage devices (OBSDs). Reed, *et al.* provide a method for strong authentication in such an environment in SCARED [10], providing an excellent platform for both standard security [7] and the intra-file security proposed in this paper.

6 Conclusions

Secure file systems and distributed storage networks currently permit encryption only on a per-file or per-directory basis. However, there are many applications that would benefit from the ability to encrypt data in smaller pieces, using different keys to permit parts of a file to be read and written by different groups of users.

This paper presents a solution to this problem, by introducing a concept called intra-file security, and provides a high-level design for implementing it in a distributed file system and on individual servers within such a file system. Intra-file security uses additional metadata to maintain information about secure segments, allowing blocks of a file to be encrypted and decrypted individually on the client. A key management system provides group management facilities that are well adapted to the hierarchical nature of access to classified materials present in organizations requiring security.

Acknowledgments

We thank Randal Burns for his feedback and advice, and Ahmed Amer for proof-reading. We also thank our shepherd, Jack Cole, for his helpful suggestions and patience.

References

[1] M. Blaze. A cryptographic file system for Unix. In *Proceedings of the First ACM Conference on Computer and Communication Security*, pages 9–15, Nov. 1993.

[2] J. Daeman. *Cipher and Hash Function Design*. PhD thesis, Katholieke Universiteit Leuven, Mar. 1995.

[3] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A cost-effective, high-bandwidth storage architecture. In *Proceedings of the 8th Interational Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 92–103, San Jose, CA, Oct. 1998.

[4] H. Gobioff, G. Gibson, and D. Tygar. Security for network attached storage devices. Technical Report TR CMU-CS-97-185, Carniege Mellon, Oct. 1997.

[5] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, pages 124–139, Dec. 1999.

[6] D. Mazières and D. Shasha. Don't trust your file server. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pages 99–104, Schloss Elmau, Germany, May 2001.

[7] E. L. Miller, D. D. E. Long, W. E. Freeman, and B. C. Reed. Strong security for network-attached storage. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, Monterey, CA, Jan. 2002.

[8] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Oct. 2001.

[9] B. C. Neumann, J. G. Steiner, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the Winter 1988 USENIX Technical Conference*, pages 191–201, Dallas, TX, 1988.

[10] B. Reed, E. Chron, R. Burns, and D. D. E. Long. Authenticating network attached storage. *IEEE Micro*, 20(1):49–57, Jan. 2000.

[11] E. Reidel, M. Kallahalla, and R. Swaminathan. A framework for evaluating storage system security. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, Monterey, CA, Jan. 2002.

[12] P. Reiher, T. Page, G. Popek, J. Cook, and S. Crocker. Truffles—secure file sharing with minimal system administrator intervention. In *Procedings of the 1993 World Conference on System Administration, Networking, and Security*, Apr. 1993.

[13] B. Schneier. *Applied Cryptography*. Wiley, New York, NY, 2nd edition, 1996.

[14] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes*. Morgan Kaufmann Publishers, 1999.

[15] E. Zadok, I. Badulescu, and A. Shender. Cryptfs: A stackable vnode level encryption file system. Technical Report CUCS-021-98, Columbia University, 1998.

Efficient Storage and Management of Environmental Information

Nabil R. Adam, Vijayalakshmi Atluri, and Songmei Yu
MSIS Department and CIMIC, Rutgers University
Newark, New Jersey 07102
{adam, atluri, songmei}@cimic.rutgers.edu

Yelena Yesha
Department of Computer Science and Electrical Engineering, UMBC
Baltimore, MD 21250
yeyesha@cs.umbc.edu

Abstract
Spatial Data warehouses pose many challenging requirements with respect to the design of the data model due to the nature of analytical operations and the nature of the views to be maintained by the spatial warehouse. The first challenge is due to the multi-dimensional nature of each dimension itself. In a traditional data warehouse the various dimensions contributing to the warehouse data are simple in nature, each having different attributes. Data models such as the star schema, fact constellation schema, snowflake schema or the multi-dimensional model, can therefore, be used to represent the traditional data warehouse. On the other hand, the different dimensions in a spatial data warehouse comprise of different types of data, each of which is multi-dimensional in nature. The current available data models are not adequate for such domains. In this paper, we propose a data model that is well suited for such domains, called the **cascaded star model** that is capable of representing multiple dimensions of a spatial data warehouse, where each dimension is multi-dimensional. The nature of the queries in such domains is different from that of traditional data warehouses (such as fly-by of a region), and therefore we propose a suitable architecture that allows specification of the queries and their visual presentation.

1. Introduction

In the area of Environmental and Earth sciences, we are concerned with collection, assimilation, cataloging and dissemination or retrieval of a vast array of environmental data. Environmental and Earth science computer systems receive their input from various types of satellite images with different resolutions captured by different sensors, models of the topography and spatial attributes of the landscape such as roads, rivers, parcels, schools, zip code areas, city streets and administrative boundaries (all exist in topographic maps), census information that describes the socio-economic and health characteristics of the population, processed digital terrain models into a new information product in the form of three-dimensional visualizations of digital terrain models projected as video ``fly-bys", and finally information transmitted (almost in real-time) from ground monitoring stations.

The system needs to provide flexible image extraction functionalities, such as hyper-spectral channel extraction, overlaying, and ad-hoc thematic coloring [4]. Such systems are intended to serve the evaluation and formulation of environmental policies by enabling users, including management and researchers to query various critical parameters such as ambient air and water quality and visualize the results in a graphical form. In addition to serving decision makers and

researchers, these systems are intended to also serve the citizens, thus, enabling any citizen of any given district or a state to look at his/her county, community, home and be able to obtain relevant information on such issues as environment, health, and infrastructure, among others. Such systems should facilitate effective knowledge discovery in a manner tailored to changing needs and abilities of users, both intellectual and technological.

Consider for example the NASA Regional Application Center (RAC) at Rutgers Center for Information Management, Integration and Connectivity (CIMIC), which is a joint project between Rutgers CIMIC, NASA Goddard Space Flight Center (GSFC) and the New Jersey Meadowlands Commission (NJMC). As a RAC, CIMIC maintains a large collection of satellite images acquired through various sources. Specifically, the CIMIC-RAC currently stores and manages satellite imagery from various sources, including:

- Direct downloads of AVHRR data from polar orbiting satellites, such as NOAA 12, NOAA 14 and NOAA 15, over the Northeast region of the US including New York and New Jersey;
- LANDSAT and RADAR data obtained from NASA archives;
- Hyper-spectral images from the Airborne Imaging Spectrometer for Applications (AISA) sensor;
- Value-added products, such as AVHRR NDVI biweekly composites from the NASA EROS data center; Aerial ortho-photographs provided by various private companies; and
- Value-added products generated by various experts.

In addition to the images from a variety of space borne satellites, other data includes ground data from continuous monitoring weather stations, and maps, reports, data sets from federal, state and local government agencies. The problem is how to efficiently manage and store this diverse type of information and how to effectively serve the diverse set of end users. In traditional domains such as banking, insurance, and retail industries data warehousing has been successfully implemented to address this problem (inmon96). In such industries, the problem of how to design and implement data warehousing has been well researched over the years and is well understood. In nontraditional domains such as the Environmental and Earth sciences, the problem of applying data warehousing technology is complex and needs further study.

2. Challenges

Environmental data warehouse is an example of a spatial data warehouse. "Spatial Data Warehouse is defined as an integrated, subject-oriented, time-variant, and nonvolatile spatial data repository for data analysis and decision making [8]." A data warehouse may use one of the data models such as the star schema, fact constellation schema, snowflake schema or the multi-dimensional model. For example, in a star schema, the data warehouse contains a central table called the fact table, comprising of the keys of each dimension, and a table for each dimension. In a spatial data warehouse, the dimensions may include both spatial and non-spatial. Spatial Data warehouses pose many challenging requirements with respect to the design of the data model due to the nature of analytical operations and the nature of the views to be maintained by the spatial warehouse.

The first challenge is due to the multi-dimensional nature of each dimension itself. In a traditional data warehouse the various dimensions contributing to the warehouse data are simple in nature, each having different attributes. On the other hand, the different dimensions in a spatial data warehouse comprise of different types of data, each of which is multi-dimensional. The various raster images such as satellite downloads, images generated from these satellite images describing various parameters including land-use, water, temperature have multiple dimensions including the geographic extent and coordinates of the image, the time and date of its capture, and resolution. Other such examples include aerial photographs. The regional maps represented as vector data also have a temporal dimension as they change over time. The streaming data collected from various sensors placed at different geographic locations that sense temperature, air quality, atmospheric pressure, water quality, dissolved oxygen, mineral contents, salinity, again have both spatial and temporal dimensions. Other dimensions include demographic data, census data, traffic patterns, and many such as these.

The second challenge is due to the nature of the queries posed to the scientific warehouses. As the queries typically involve accessing multiple dimensions, each of which in itself is multi-dimensional. We illustrate this with the following examples:

Example 1: A user may want to look at the changes in the vegetation pattern over a certain region during the past 10 years, and see their effect on the regional maps over that time period. This involves layering the images representing the vegetation patterns with those of the maps whose time intervals of validity overlap, and then traverse along this temporal dimension with the overlaid image. In the traditional data warehouse sense, this amount to first constructing two data cubes along the time dimensions for each of the vegetation images and maps, and then fusing these two cubes into one. One may envision fusing of multiple cubes. For example, if the user also wants to observe the changes in the surface water, population, etc., due the changes in the vegetation pattern over the years, fusion of such multiple cubes is needed.

Example 2: Another user may want to simulate a fly-by over a certain region staring with a specific point and elevation, and traverse the region on a specific path with reducing elevation levels at a certain speed, and reaching a destination, effectively traversing a 3-dimensional trajectory. This query involves retrieving images that span adjacent regions that overlap the spatial trajectory, but with increasing resolution levels to simulate the effect of reduced elevation level. Another important aspect of serving such queries additionally requires controlling the speed at which they are displayed to match the desired velocity of the fly-by.

3. Spatial Database System Architecture

The ingestion, processing and storing of satellite images in CIMIC is done as shown in Figure 1. Images are downloaded from NOAA satellites with the Quorum HRPT antenna and receiver systems. Once a day the new raw image files are moved to oversized hard drives on a UNIX HP platform. At the same time, a new elements.dat file with ephemeris data is captured through the web and placed in the PC running the QTrack ingest software, which assures that images ingested later on will have updated orbital elements information and require less navigational correction.

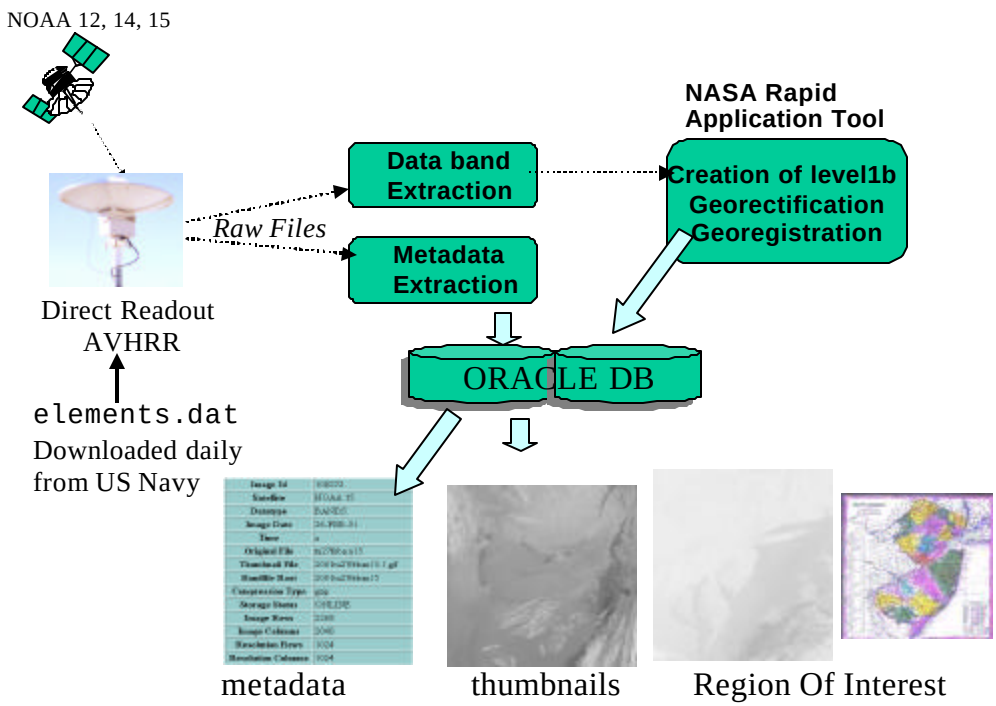


Figure 1: Preprocessing and Ingesting of Satellite Images

On the HP platform, raw files are fist classified by size. Files less than 20mb are automatically eliminated, and the remaining raw files are converted to level-1b by a quick-ingest routine, and then compressed. Level-1b files then go through the remap routine where images are clipped to a specific area of interest (New Jersey and surroundings) and projected to the Mercator projection. The resulting remap files are saved in an internal format (RAT format) and as bitmap files. These bitmap files are then classified using normalized regression routine, which employs a tool developed by NEC. Specifically, images with high regression coefficient (0.80 or greater) are classified as cloud free for the region of interest and flagged as so in the database. The RAT format files that emerge from the remap tool are used to create NDVI's. These NDVI's populate the database and become available to users through the web, and bi-weekly collection of NDVI's are made into a single NDVI images composite and are also available through the web. Due to the limited use of DBMS extenders for handling spatial data, we have implemented the database in two separate modules: One the relational DBMS to store metadata and thumbnail of images, and another a spatial data/flat file for images. Image files are tied with the DBMS by linking the image-id in the database with individual image files. The metadata of the images is maintained by an Oracle database through which image thumbnail images can be obtained. These images are indexed using an *SS-Tree* for enhancing the response time for the queries and insertions.

Interfaces are provided to querying the database based on time of capture, particular satellite or sensor instrument, type of image such as raw, composites, NDVI, water, temperature, etc. Essentially, users are provided with the image-ids, and the actual image is retrieved by clicking on the relevant image-id. Currently, it does not provide powerful capabilities to let users

perform complex queries for advanced data analysis, such as trend or pattern analysis. In addition, no visual display tools are available to allow users to view image pattern changes over certain period of the range queries displayed with a speed specified by the users, nor capabilities to handle queries that simulate a fly-by over certain region as described in Examples 1 and 2.

Currently it uses ArcIMS from ESRI to process the image files (in .shp format), including layering the images, populating the metadata associated with the images, coloring, and composing fly-bys. These are then published on the web so that users can view them, zoom-in/out, move in different directions (north, south, east, west), or get associated metadata by clicking on a specific place. However, this is accomplished manually only for a pre-specified set of queries. Our goal is to accommodate ad-hoc queries by employing a data warehouse. As a result, for example, the above-mentioned fly-bys can be automatically generated upon users' request.

4. The Spatial Data Warehouse System Architecture

Our system comprises of a friendly geographic user interface, a powerful query processing engine that is capable of supporting various OLAP operations, an output rendering engine, and an spatial data warehouse, as shown in figure 2. Our data warehouse is based on the *cascaded star* model, described in section 6.

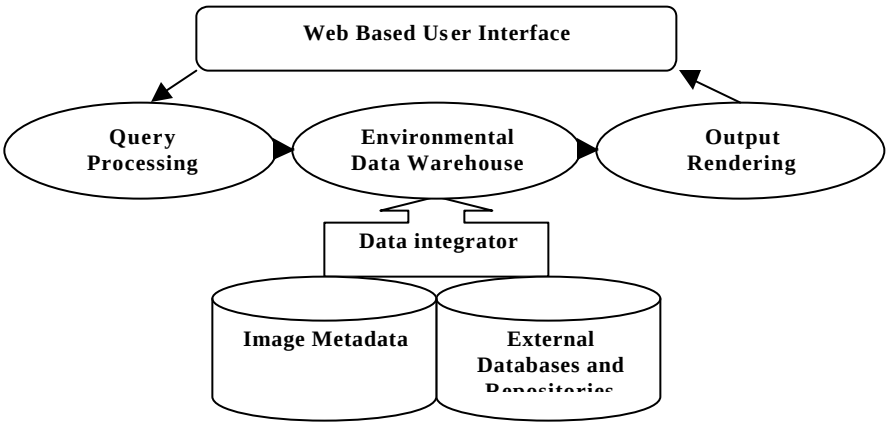


Figure 2: System Architecture

The data from different repositories, such as metadata databases, image database, databases of real-time streaming data from environmental sensors, etc., are first extracted, validated, transformed and then finally integrated, before loading into the warehouse. The data in the warehouse is periodically refreshed to reflect updates at the sources and purged from the warehouse, perhaps onto slower archival storage [10].

In general, the reason one builds a data warehouse is to construct data in a structured way and to allow pre-processing so that users can turn the data into useful knowledge quickly. Operational databases maintain state information, while data warehouses typically maintain historical information, and as a result, data warehouses tend to be very large and grow over

time. Hence, the size of the data warehouse and the complexity of queries can cause queries process to take very long to complete, which is unacceptable in most decision support system environments. Also, a major performance challenge for implementing query processing and output representation is how we construct data warehouse in an efficient way.

4.1 Constructing an Efficient Data Warehouse

There are many ways to achieve data warehouse performance goals. Query optimizations and query evaluation technique can be enhanced to handle aggregations better, or using different indexing strategies like bit-mapped indexes and join indexes, etc. We consider implementing our GIS warehouse in the following two specific aspects to facilitate construction of the efficient data warehouse.

One commonly used technique is to selectively materialize/pre-compute frequently used queries. If we can do this pre-computation effectively and efficiently, then we can store many frequently accessed historical results in the data warehouse combined with different time periods, different resolutions, different aggregations, and different views, etc, at users' interests. In this way, the output processing can be achieved very fast, and sometimes automatically without any more computation efforts.

Firstly, let us look at the pre-computation for non-spatial data that are stored in RDBMS and are associated with spatial data. Picking the right set of queries to materialize is a nontrivial task. For example, we may want to materialize a query that is relatively infrequently used if it helps us answer many other queries quickly. We adopt the linear cost model from [8], where the data are stored in multi-dimensional data cubes, and each cell of the data cube is a view consisting of an aggregation of interest. The values of many of these cells are dependent on the values of other cells in the data cube. One common and powerful query optimization technique is to materialize some or all of these cells rather than compute them from raw data each time. A lattice framework is used to express dependencies among different cells in the total or partial order, and a greedy algorithm that works off this lattice determines a good set of cells to materialize [9]. We all know that dimensions of a data cube consist of more than one attribute, and the dimensions are organized as hierarchies of these attributes. For a simple example, the time dimension can be organized into the hierarchy: day, week, month, and year as follows:

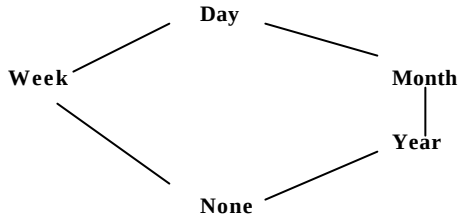


Figure 3: Sample Time Hierarchy

In the presence of above hierarchy, the dependency relationship is obviously seen. Consider a query that groups on the time dimension only, and we can have the following three queries possible: (day), (month), (year), each of which groups at a different granularity of the time dimension, also if we have total available for by month, we can use the results to compute the

total grouped by year. Generally we selectively materialize the data cube based on query dependencies introduced by the conception of hierarchies.

Secondly, it is also essential to pre-process spatial data efficiently, which are more complicated than computing non-spatial data. For example, we may pre-process digital maps at different resolution levels and store them in the data warehouse, and users can combine them randomly to stimulate a fly-by, or pre-overlay the images representing the vegetation patterns with those of the maps having the same time intervals of validity, or pre-group a multi-color coded map to emphasize a particular category, or pre-interpolate spatial data over a large area which refers to the process of deriving elevation data for points where no data samples have been taken, etc. There is a big challenge for our project since our pre-processing is based on users' most frequent access interests that have to be updated frequently to meet changes.

Another challenge is that the above partial or total order relationship may not be suitable for spatial data dependency. For example, there is no dependency relationship among resolutions, and we can't compute high-level resolution based on low-level resolution or vice versa, or we can't overlay two images based on another overlaid image. Finding a dependency relationship among spatial data to avoid processing every raw image from scratch is our next step.

Another technique is to construct our data warehouse model in a different way that is an extension of the star schema, in which each dimension itself has a star schema of its own. We will explore this in detail in the following section.

4.2 The User Interface, Query Processing and Output Rendering Engines

A web based high-level user interface to a GIS must provide users with the necessary tools to store, retrieve, and analyze data so that they can perform their application-specific functions. More importantly, it is used to perform complex data analysis from the data warehouse without writing programs and should be comprehensive enough to let users get detailed analysis results and knowledge.

Moreover, after the translated SQL queries are processed in the data warehouse, an output will present multi-dimensional views of data to various front-end tools through different output processing engines. For example, OLAP servers can execute all OLAP operations, such as roll-up, drill-down, dicing and slicing, and generate results for data analysis and reporting, decision making strategies and advanced data mining. At the same time, users could require the data representation as the generation of a fly-by video with a trajectory, elevation and velocity.

When a spatial database is to be used interactively, graphical presentation of spatial data types (SDT) values in query results is essential. It is also important to enter SDT values to be used as "constants" in queries via a graphical input interface. The goal of querying is in general to obtain a "tailored" picture of the space represented in the database, which means that the information to be retrieved is often not the result of a single query but rather a combination of several queries. For example, in GIS application, the user may want to see a map built by graphically overlaying the results of several queries. Therefore, a user interface for output presentation should have at least two sub-windows: (1) a text window for displaying the textual

representation of a collection of objects, containing the metadata or alphanumeric attributes of each spatial object, (2) a graphical window containing the overlay of the graphical representations of spatial data of several object classes or query results, which could be a generation of a fly-by video. We will consider implementing our system in this way in the near future.

The query engine translates the user inputs as SQL queries that will be inserted into data warehouses for further processing. The output representation engine is dealing with data representation using existing software such as PIT and IDRISI or newly developed applications. This part is mainly complicated by users' requirements because there are a lot of decision-support queries that are much more complex than OLTP queries and make heavy use of aggregation, and this is basically OLAP operations. Besides this, most users need some specific visualization results such as fly-by over a certain region staring with a specific point and elevation, and traverse the region on a specific path with reducing elevation levels at a certain speed, and reaching a destination, effectively traversing a 3-dimensional trajectory, or a fly-by over a certain time period for vegetation pattern change within New Jersey area, which is a process of image manipulation and representation.

5. Traditional Data Warehouse Models

A number of data models have been proposed to conceptually model the multi-dimensional data maintained in the warehouse. These include the star schema, the snowflake schema, and the fact constellation schema. Since our data model, the cascaded star model, is an extension of the star model, in the following, we present these three models with examples, and bring out the limitations of these models in representing the data in our spatial data warehouse.

5.1 The Star Schema

Perhaps, star schema, first introduced by Ralph Kimball, is the earliest schema used to model the data warehouse implemented as a relational databases. In this schema, the data warehouse contains a large central table (fact table) containing the bulk of data (dimensions) with no redundancy, and a set of smaller attendant tables (dimension tables) with one for each dimension. The schema graph resembles a starburst, with the dimension tables displayed in a radial pattern around the central fact table, as shown in Figure 4, where A is the fact table, and b, c, d, e and f are dimensions and represented by dimensional tables.

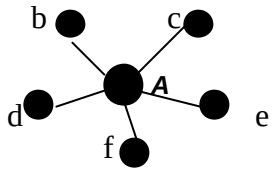


Figure 4: The Star Model

Note that in the star schema, only one dimension table represents each dimension, and each dimension table contains a set of attributes and joins with fact table by common keys when implemented as a relational database. Moreover, the attributes within a dimension table may form either a hierarchy (total order) or a lattice (partial order). Currently, most traditional data

warehouses use a star schema to represent the multi-dimensional data model as it provides strong support for OLAP operations.

To illustrate, in the following, we provide an example of the implementation in star schema [8]. Suppose the multi-dimensional data for the weather in northeast region in USA consists of four dimensions: temperature, precipitation, time, and region_name, and three measures: region_map, area, and count, where region_map is a spatial measure which represents a collection of spatial pointers pointing to corresponding regions, area is a numerical measure which represents the sum of the total areas of the corresponding spatial objects, and count is a numerical measure which represents the total number of base regions accumulated in the corresponding cell.

The following figure illustrates the implementation for a star model in this case:

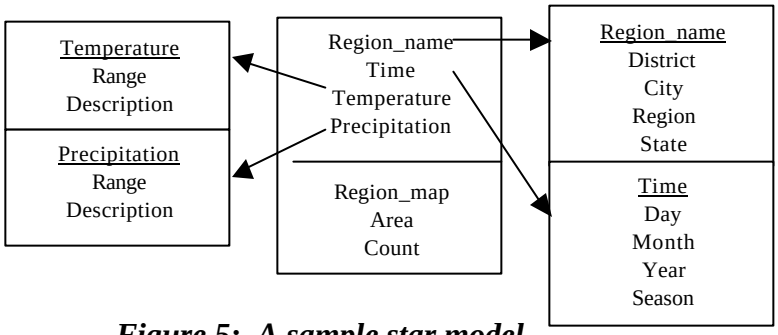


Figure 5: A sample star model

The following tables show some sample data set that maybe collected from a number of weather districts tested in northeast of USA.

Region_name	Time	Temperature	Precipitation	…
A111	02/23/01	33	1.4	…
B111	02/24/01	41	1.5	…
…	…	…	…	…

Region_name	District	City	Region	State
A111	A	Flushing	111	NY
B111	B	Edison	111	NJ
…	…	…	…	…

Time	Day	Month	Year	Season
02/23/01	23	February	2001	Winter
02/24/01	24	February	2001	Winter
…	…	…	…	…

Temperature	Range	Description
33	11	Chilly
41	12	Mild cold
…	…	…

Precipitation	Range	Description
1.4	21	Middle
1.5	22	Middle
…	…	…

From this sample, we can see that a star model consists of a fact table with multiple dimension tables, and the fact table joins the dimension tables with different keys. In this example, all attributes in each dimension table are only one-dimensional and can be expressed completely in one table. Our question is: if some or all of the attributes in the dimension tables are also multi-dimensional, i.e., one attribute in one dimension table has multiple attributes associated with it, how can we implement it in this model? The answer is impossible.

5.2 The Snowflake Schema

Snowflake schemas provide a refinement of star schemas where the dimensional hierarchy is explicitly represented by normalizing the dimension tables, and therefore further splitting the data into additional tables (see Figure 6). Such a table is easy to maintain and saves storage space because a large dimension table can become enormous when the dimensional structure is included as columns.

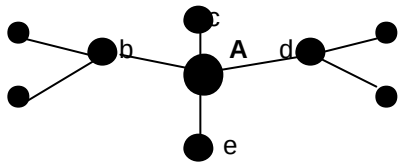


Figure 6: The Snowflake Model

However, only some dimensional tables are normalized and this normalization reduces the effectiveness of browsing since more joins will be needed to execute a query. When applied to spatial attributes for each dimension table in our case, it is obviously not well suited.

5.3 The Fact Constellation Schema

Sophisticated applications may require multiple fact tables to share dimension tables. The dimensions of this expanded star schema can be normalized into a snowflake schema. These multiple fact tables can separate the detail and the aggregated values instead of maintaining a single and huge fact table, which may speed the queries processing. See Figure 7 for this schema, where fact table A and B share the dimensions h and i.

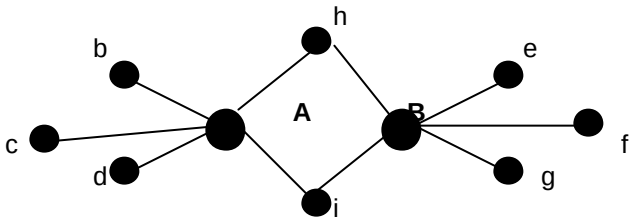


Figure 7: The Fact Constellation Model

However, there are some disadvantages of using the fact constellation schema. For example, for data warehouse with high cardinality, i.e. high number of hierarchy, numerous fact tables must be created, which increase the complexity of the design. Furthermore, for spatial oriented attributes for each dimension table, only one dimension table is not enough for holding the properties of each attribute.

6. The Cascaded Star Model

In this section, we present an outline of our spatial data warehouse model, called the **cascaded star schema**, which is an extension of the star schema, where each dimension itself has a star schema of its own. There are a number of research studies in the area of spatial data warehouses (see the reference list). The work proposed by Han et al. is closely related to our work. Han et al. [8,9] study the problems associated with the design and construction of spatial data cubes. It distinguishes the various dimensions in the spatial data warehouse as non-spatial, spatial-to-non-spatial, spatial-to-spatial, based on how they transform when that dimension is generalized. They provide how the various operations such as roll-up, drill-down, slicing and dicing, and pivot can be carried out. While we recognize that each spatial dimension in a data warehouse in itself is multi-dimensional and argue that the data warehouse model need to be enhanced to handle this. The cascades star schema is shown in Figure 8, where A is the fact table, and b, c, d, e and f are dimensions that are also multi-dimensional.

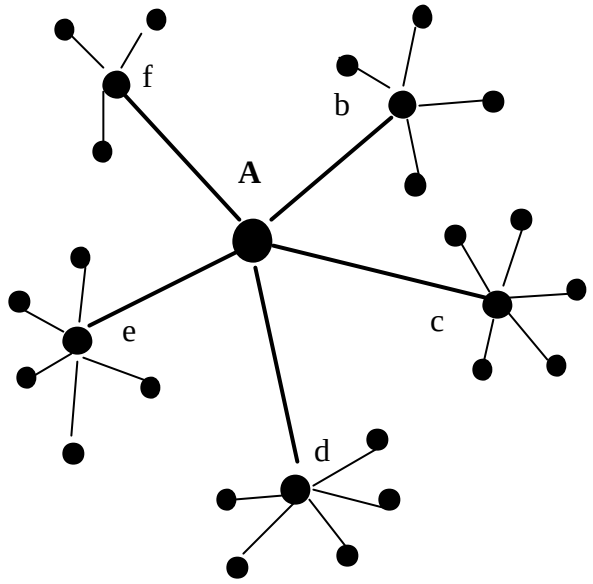


Figure 8: The Cascaded Star Model

The multi-dimensional nature of each dimension is illustrated with an example in figure 9. In here, the fact table comprises of the various dimensions of the spatial data, which include land-use, temperature, water and vector maps. As can be seen, each of these dimensions in turn is multi-dimensional, represented as a star. To illustrate, the land-use dimension comprises of a fact table of its own with dimensions time, spatial and attributes, where the time dimension is comprised of attributes year, date and time of capture of the image; the spatial dimension is comprised of the x, y coordinates of the lower left hand and corner and the upper right hand

corner of the region covered by the image, and the resolution; the attributes dimension is comprised of the amount of vegetation, developed, barren, forested upland, etc. in the image. Similar to land-use, as can be seen from the figure, themes and water dimensions are also multi-dimensional in nature.

In the paper, we will present our detailed data model, and introduce the necessary primitives that enable the evaluation of different queries. We will also discuss what the different warehouse operations such as drill-down, roll-up, mean in the semantic sense in the cascaded star schema, and show how they can be carried out. We will present the architecture of our prototype and the guidelines for implementation.

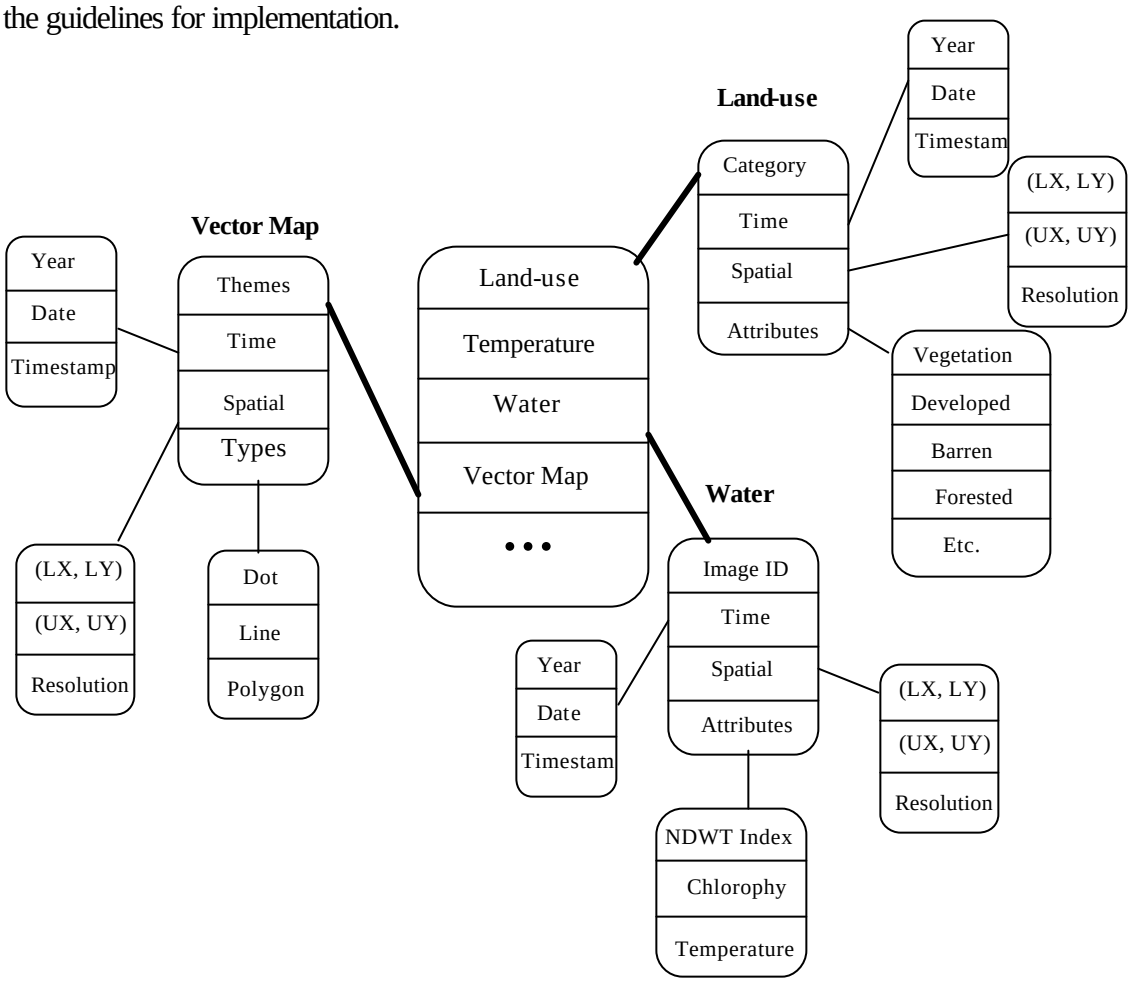


Figure 9: A Sample Cascaded Star Model

The following tables show some examples of these dimensions:

Fact table:

Land_use	Temperature	Water	Vector Map	…
Abc	44	221	111	…
…	…	…	…	…

One dimension table: "Vector Map"

Vector_map	Themes	Time	Spatial	Types
111	New Jersey	01	A	
…	…	…	…	…

Another dimension table for an attribute "Time" in "Vector Map":

Time	Year	Date	Timestamp
01	2000	3/23/00	12:00am
…	…	…	…

In the above example, we can see that a fact table is joined with several dimension tables as in the star model, and each attribute in the dimension tables is self multi-dimensional with another dimension table joined with it. In this easy way, we implement a cascaded star model for each multi-dimensional attribute in the dimension tables, which explicitly provides support for attribute hierarchies. However, the previous star schema cannot accomplish such multi-dimensional attribute structures in a single way.

We want to address the difference between a cascaded star model and a snowflake model. Someone may get the false impression at first sight that there is no big difference between these two models since they both have multiple extensions for some spatial dimensions. However, a snowflake model just normalizes some dimensions to reduce a big dimension table for easy maintenance and storage saving, whereas a cascaded star model claims each dimension itself is multi-dimensional by the nature.

6.1 OLAP Operations on the Cascaded Star Model

Now let us examine some popular OLAP operations, i.e., roll-up, and drill-down, slicing and dicing, and pivoting, and analyze how they are performed in the spatial data cube we constructed in a cascaded star model. OLAP are traditional data warehouse operations that provide users to view data from different perspectives, hence, OLAP support user-friendly environment for data analysis and prepare for advanced data mining process. In the system architecture we proposed, it is part of the output rendering engine.

These operations have been discussed intensively in the traditional data warehouse and spatial data cube in star model [7]. Our concentration is that how they can be efficiently operated in the star cascaded model with selectively materialization, which means aggregating and generalizing data from multi-dimensional attribute tables. Consider the example 1 we mentioned above. A user may want to look at the changes in the vegetation pattern over a certain region during the past 10 years, and see their effect on the regional maps over that time period. This query involves two very commonly used querying operations of OLAP: "drill-down" and "roll-up". We constructed the time hierarchy with a partial order in the above and they underlie these two operations. Drill-down is the process of viewing data at progressively more detailed levels, for example, a user drills down by first looking at the vegetation pattern per year and then comparing the vegetation pattern by specific month within different years. Roll-up is just the opposite, which is the process of viewing data in progressively less detail. In roll-up, a user starts with the vegetation pattern on a given month, then looks at the total pattern in that year, and finally, compares the patterns among 10 years. With selective pre-computation of certain

data cells in the multi-dimensional data cube, such as vegetation pattern for each month within each year, we can easily process this query.

7. Related Work

Research in data warehousing is a relatively new area. In the following we review the research contributions as well as the prototypes that are most relevant to our work. Han et al. [8,9] proposes a spatial data warehouse model in which both spatial and non-spatial dimensions and measures exist. It proposes spatial data cube construction based on approximation and selective pre-computation spatial OLAP operations, such as merge of a number of spatially connected regions. The pre-computation involves spatial region merge, spatial map overlay, spatial join, and intersection between lines and regions.

Microsoft TerraServer [2] stores aerial, satellite, and topographic images of the earth in a database available via the Internet, where the users are provided intuitive spatial and text interfaces to the data. Basically terabytes of "Internet unfriendly" geo-spatial images are scrubbed and edited into hundreds of millions of "Internet friendly" image tiles and loaded into a data warehouse. The TerraServer adopts a "thin-client and fat-server" model, which consists of three tiers: the client tier, the application logic tier, and the database system tier. Users can search the data warehouse by coordinates and place names, and can easily view the images with different resolutions by simply clicking on it. The application logic responds to the HTTP requests and interacts with the back end database to fetch the results. The database is a SQL server 7.0 RDBMS containing all images and meta-data of images that are pre-processed and stored, for example, all levels of the image pyramid (7 is maximum) are pre-computed and stored. However, this system does not provide powerful and comprehensive image pre-processing tools such as spatial OLAP for advanced spatial data analysis. Moreover, the RDBMS integration with image repository has inherent problems, as SQL server 7 stores imagery in JPEG or GIF format which does not have much flexibility in handling spatial data.

However, none of the prior researchers recognize that each dimension in a data warehouse in itself is multi-dimensional. As a result, much of the work in spatial data warehousing is based on the star model. However, this work does not address the issue of the nature of spatial data warehouse.

8. Conclusions and Future Research

In this paper we focused on the problem of applying data warehousing technology in order to efficiently manage, store as well as effectively serve users of environmental and earth science information centers. An example of such centers is the Regional Application Center, which is collaboration between NASA, Rutgers CIMIC and New Jersey Meadowlands Commission (NJMC). In this paper, we recognize that environmental data warehouse differs from that of a traditional data warehouse in that, each dimension in itself is multi-dimensional in nature. We have proposed a new data model, called the cascaded star model to accommodate this. In this paper, we have provided a limited treatment to the OLAP operations. Our future work includes formalizing the necessary primitives that enable the specification and execution of queries, and the semantics of various warehouse operations including, drill-down and roll-up and the evaluation of these operations.

9. Acknowledgment

This work is supported in part by the Meadowlands Environmental Research Institute, Rutgers University.

References

[1] Nabil R. Adam, Aryya Gangopadhyay, "Database Issues in Geographic Information Systems", Kluwer Academic Publishers, 1st edition, 1997.

[2] Tom Barclay, Jim Gray and Don Slutz, "Microsoft TerraServer: a spatial data warehouse," Proceedings of the 2000 ACM SIGMOD on Management of data, pages 307-318.

[3] Peter Baumann, "Web-enabled Raster GIS Services for Large Image and Map Databases," Proceedings of the ACM DEXA2001, pages 870 - 874.

[4] Wendolin Bosques, Ricardo Rodriguez, Angelica Rondon and Ramon Vasquez, "A Spatial Data Retrieval and Image Processing Expert System for the World Wide Web," 21st International Conference on Computers and Industrial Engineering, 1997, pages 433-436.

[5] Ron Briggs, "NSDI Demonstration Project: Final Report", http://www.bruton.utdallas.edu/research/usgs/usgsframe.html

[6] Volker Coors, Volker Jung, "Using VRML as an Interface to the 3D Data Warehouse", *Proceedings of the third symposium on Virtual reality modeling language*, 1998, Page 121-129.

[7] Martin Ester, Hans-Peter Kriegel, Jorg Sabder, "Knowledge Discovery in Spatial Databases", Invited Paper at 23rd German Conf. on Artificial Intelligence (KI '99), Bonn, Germany, 1999.

[8] Jiawei Han, Nebojsa Stefanovic, and Krzysztof Koperski, "Object-Based Selective Materialization for Efficient Implementation of Spatial Data Cubes ", IEEE Transactions on Knowledge and Data Engineering, 12(6): 938-958, 2000.

[9] J. Han, N. Stefanovic, and K. Koperski, "Selective Materialization: An Efficient Method for Spatial Data Cube Construction", Proc. 1998 Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'98), Melbourne, Australia, April 1998, pp.144-158.

[10] Venky Harinarayan, Anand Rajaraman, and Jefferey D. Ullman, "Implementing Data Cubes Efficiently", Proceedings of ACM SIGMOD Int'l. Conf. on Management of Data, Montreal, Canada, June 1996.

[11] R. Holowczak, N. Adam, F. Artigas, and I. Bora, "Data Warehousing for Environmental Digital Libraries." To appear in Communications of the ACM, 2002.

[12] N. Widmann, P. Baumann, "Towards Comprehensive Database Support for Geoscientific Raster Data," Proceedings of ACM-GIS'97, Las Vegas/USA, November 1997.

Indexing and selection of data items in huge data sets by constructing and accessing tag collections

Sébastien Ponce
CERN *
LHCb Experiment
sebastien.ponce@cern.ch
tel +1-41-22-767-2143

Pere Mato Vila
CERN *
LHCb Experiment
pere.mato@cern.ch
tel +1-41-22-767-8696

Roger D. Hersch
Ecole Polytechnique
Fédérale de Lausanne †
Computer Science Department
RD.Hersch@epfl.ch
tel +1-41-21-693-4357
fax +1-41-21-693-6680

Abstract

We present here a new way of indexing and retrieving data in huge datasets having a high dimensionality. The proposed method speeds up the selecting process by replacing scans of the whole data by scans of matching data. It makes use of two levels of catalogs that allow efficient data preselections. First level catalogs only contain a small subset of the data items selected according to given criteria. The first level catalogs allow to carry out queries and to preselect items. Then, a refined query can be carried out on the preselected data items within the full dataset. A second level catalog maintains the list of existing first level catalogs and the type and kind of data items they are storing.

We established a mathematical model of our indexing technique and show that it considerably speeds up the access to LHCb experiment event data at CERN (European Laboratory for Particle Physics).

1 Introduction

Indexing and data selection in a huge data set having a high index dimensionality is one of the key issue in the domain of data management. Recent papers on the subject address this

*CH-1211 Geneva 23, Switzerland
†CH-1015 Lausanne, Switzerland

problem in specific cases such as spatial databases [6, 5, 7], similarity searches [5, 1, 8] or string matching [4] but do not offer global solutions. Moreover, existing methods are outperformed on average by a simple sequential scan when the number of dimensions is larger than approximately ten[13].

On the other hand, the variety of useful selection criteria on a given set of data is far from being infinite. They can usually be reduced to a small number of indexes, say 20 to 30 maximum (which is already a very high dimension). Thus, from all values contained in a data item (tens of thousands in some cases), only this very reduced subset of 20 to 30 values is relevant for the selection criteria.

This property is used to define a new indexing method based on two levels of catalogs. This method greatly speeds up the linear selecting process by replacing scans of the whole data by scans of matching data. Data is efficiently selected using both server side and client side preselections and the power of the SQL language.

Section 2 presents the context of this work, i.e. the LHCb experiment at CERN and its requirements in terms of data indexing and retrieval. Section 3 presents search results in the domain of data indexing and emphasizes their respective strengths and weaknesses. Section 4 presents the proposed indexing schema and shows how it can be used efficiently for data retrieval. Section 5 evaluates the performance of the new indexing method compared to sequential scan[1]. Section 6 draws the conclusions.

2 Context

The work presented here is based on studies made at CERN (European Laboratory for Particle Physics) in the context of the LHCb [10] experiment. We present here briefly the problem and the requirements we had.

2.1 The LHCb experiment

LHCb [10] is the name of one of the future Large Hadron Collider (LHC) experiments. Its primary goal is the study of the so called CP Violation [11]. This physical theory suggests that, in the world of subatomic particles, the image of a particle in a mirror does not behave like the particle itself [9]. One of the fundamental grounds of this effect is the existence of the bottom-quark and its cousin the top-quark. This is precisely this bottom-quark, under the form of the B-meson that the LHCb experiment intends to study. The only way to produce particles like this meson is to collide other high energy particles (accelerated protons in the case of LHC). This collision will produce hundreds of new particles among which the physicists will try to detect B-mesons and to measure their parameters and behavior (especially the way they decay).

[1] Sequential scan is besides our method the only method which, to our knowledge, fulfills our requirements

2.2 Some figures

LHC will let bunches of protons collide every 25 ns, i.e. at a frequency of 40 MHz. Such a collision is called an event and creates lots of particles (some hundreds). The different detectors constituting LHCb are able to detect all created particles and to specify their energy and momentum. The global output is about 1 MB of data per event across 950000 channels. This yields 40 TB of data every second !

Most of this data will not be stored since more than 99,9999% of it is not interesting. Actually, the detector has a four level trigger system that allows a reduction of the data rate from 40 TB/s to 20 MB/s per second, which is two million times less. This factor is due to both a reduction of the event size to the order of 100 KB and to a reduction of the event rate to 200 Hz. Assuming that the LHC will run 24 hours a day and 7 days a week, LHCb will produce an order of 10^{10} events per year, which is one petabyte (1 PB = 10^{15} bytes) in term of data size.

Table 1 summarizes the figures concerning the data being saved, indexed and later retrieved by physicists for analysis.

Size of a data item	100 to 200 KB
Nb of items	10^9 to 10^{10} per year
Global size of the database	$\sim 10^{15}$ bytes = 1 PB per year
Data items input rate	200 Hz
Data input rate	20 to 40 MB/s

Table 1: Figures concerning LHCb data

2.3 Data selections

The analysis by physicists of the LHCb data is rather specific. It is mainly based on an iterative process consisting in selecting some data items (typically in the order of 10^6) with rather complicated selection criteria, downloading the items, running some computation on them and modifying the selection criteria. A criterion may for example make use of the energy of the event, of the types of particles involved or of the number of decays. The number of iterations is rather small (in the order of 10) but the selection of the data still appears to be the key of the physics analysis.

Another issue is the number of indexes that a given criterion uses. This is typically in the range of 10 to 30 parameters with a mixture of numeric, boolean and strings. These indexes are not always the same for all criteria but a few number of criterion types can be defined (less than 10) for which the set of parameters is fixed. Due to the high dimensionality of the event data (10 to 30 indexes), up to now, at CERN, the only data selection algorithm was a linear scan of the whole dataset.

3 Related Work

There are not many research approaches addressing the issue of indexing generic data in a high dimension space. Weber et al [13] show that there exists a dimension over which any algorithm is outperformed by a sequential scan. Experimentations show that the sequential scan outperforms the best known algorithms such as X-trees[2] and SR-Trees[5] for even a moderate dimensionality (i.e. ~ 10).

These two algorithms are based on data partitioning methods. The ancestor of the data partitioning method is the R-tree method [3] which was further developed under the form of R^*-Trees [6]. However, these data partitioning methods perform poorly as dimensionality increases due to large overlaps in the partitions they define. This is due to exponential increase of the volume of a partition when the number of dimensions grows.

The SR-Tree method tries to overcome this problem by defining a new partition schema, where regions are defined as an intersection of a sphere and a rectangle. The X-Tree method, on the other side tries to introduce a new organization of the partition tree which uses a split-algorithm minimizing overlaps. The results are good at low and moderate dimensions but are outperformed by a sequential scan for dimensions larger than 10.

4 A two level indexing schema

The aim of our proposed schema is to allow most of the selection to be carried out using catalogs (tag collections) that contain only a part of the data items and, for each item, only a subset of its values (a tag). Several catalogs are built, each for a different type of query. This allows to perform a very efficient preselection of the items before accessing the real data items.

4.1 Tags

A tag is a subset of a data item comprising several parameters plus a pointer on this data item. A pointer is simply the information needed to find and retrieve the data item, be it a regular pointer (memory address), a file name, an URL or something else.

A tag contains the few values (also called parameters) of the data item that are used as selection criteria. For a given criterion, or even a given type of criterion, the number of tag values is small (10 to 30) which results in a tag size of 10 to 200 bytes. For example, in the case of some physics events, one may want to include in the tag the energy, the nature of the event and the number of particles involved.

Several types of tags can be defined, with different sizes and contents, even for the same data item. Different tags will point to different subsets of the data items and correspond to different criteria.

Tags are small, well structured objects that can be easily stored in a relational database.

Thus, they can be searched using the power of SQL-like languages. The storage of tags in a relational database is trivial : each type of tag is stored in a different table, whose columns are the different values included in the tag plus one for the pointer to the real data item. The data item itself does not need to be part of a database.

Tags will be used to make preselections without loading the data items, which reduces the amount of loaded data by a factor of 10^3 in the case of LHCb.

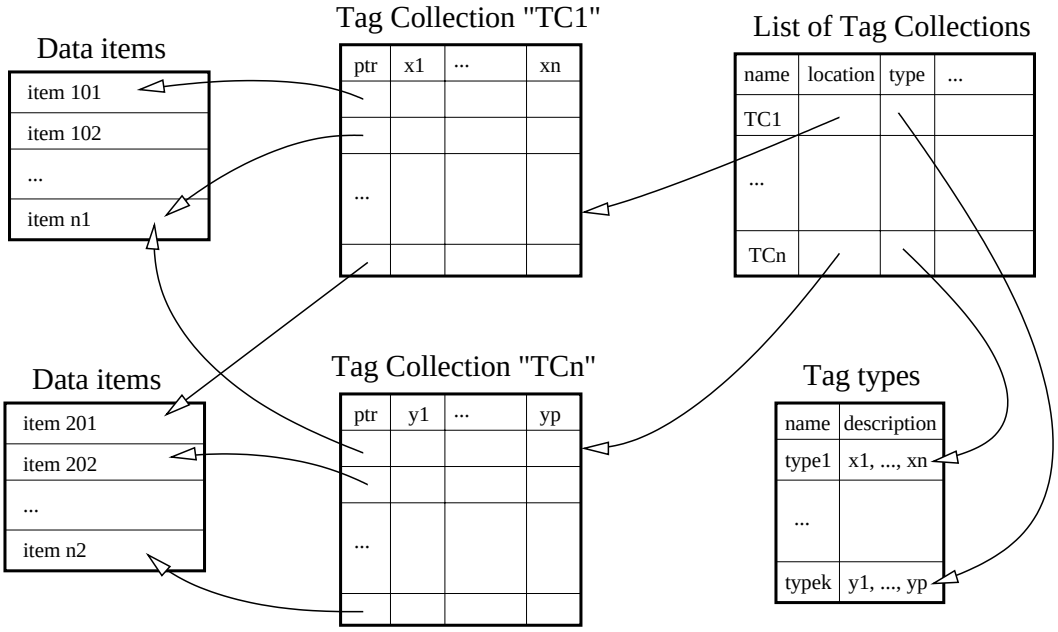


Figure 1: Structure of the tag collections

4.2 Tag collections

As explained above, tags are subsets of data items. A tag collection is a set of tags, all of the same type. It corresponds to a set of data items but with only a subset of the data items values. The values themselves fulfill some criteria, such as being in the interval between a minimal and a maximal value. Thus, two different tag collections may correspond to two different subsets of data items, even if they use the same set of values (type of tags). These subsets may of course overlap.

Tag collections are stored in a relational database as a table, where each line is a tag and columns correspond to the values contained in the tags (Fig. 1). The tag collections form a list of tag collections, each with each associated tag type.

Since tag collections only contain tags for a given subset of the data items, they act as a first preselection on data. For example, in the LHCb experiment, a collection of tags is in the order of 10^5 smaller than the database, i.e. around 10 GB. A factor 10^3 is due to the tag size (section 4.1) and another 10^2 factor comes from the fact that, on average, less than

1% of the data items have a tag in a given collection, i.e. tags whose values are within the predefined ranges associated to that collection. Thus, a collection has typically 10^7 to 10^8 entries.

Collections can be defined by any user or group of users who wants to be able to use a new selection criterion. The creation of a new collection may either require a scan of the full set of data items or is extracted as a subset from another collection. Scanning the full set of data items is time consuming but will be far less frequent than the selection of data items. We expect that there will only be 10 to 20 "base" tag collections in LHCb. All other collections will be subsets of base tag collections.

4.3 Selection process

By selecting tags in tag collections instead of selecting directly data items, there is an immediate gain. Only data items of interest are loaded instead of loading all items for each selection.

This is specially interesting in the case that data items are not located in a database but in regular files and loading a data item requires accessing a file containing many items. With a pointer to the data item within the file, the item of interest is directly accessed and loaded. Such a strategy of storing the actual data in regular files may actually be applied to many problems since database management systems cannot handle petabytes of data easily.

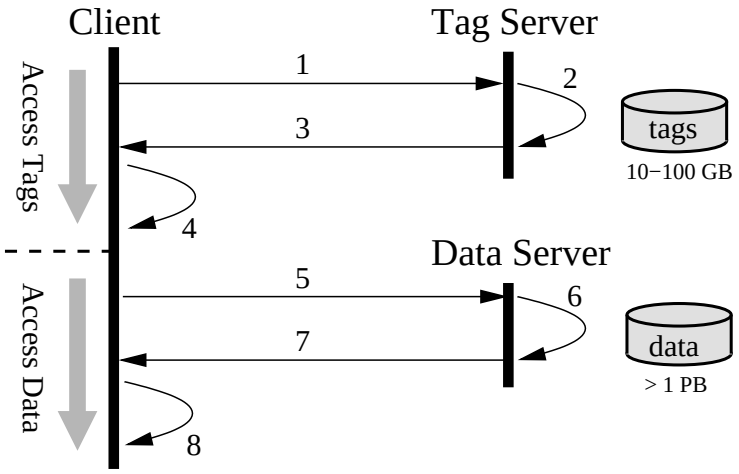


Figure 2: Data selection process

Furthermore, the 2-level indexing schema presented here offers a very powerful and flexible way of applying various preselections allowing to reduce both the amount of accessed data and the network traffic. The complete selection process is shown in Figure 2.

The steps involved in the selection process are the following :

1. The client selects a tag collection and sends a SQL query to be applied on tags from

this collection. The usage of a specific collection is actually a first preselection made by the physicist.

2. The query is processed on the server side.
3. Only matching tags are sent back. This minimizes the network load.
4. A second selection may be applied on the client side, for example for queries that cannot be formulated in SQL and which require a procedural language such as C^{++}.
5. Once the selection on tags is complete, requests for the corresponding data items are sent to the data server.
6. Data items are retrieved (from files, in the LHCb experiment).
7. Retrieved data items are sent to the client.
8. A last selection may be performed on the full data items, in the case that some information was missing in the tags which did not allow to perform this more narrow selection in a previous step.

Note that the separation between client and servers (a tag server and a data item server) on Figure 2 allows for example to replicate the tag server while keeping the data item server at a single location .

5 Performance evaluation

Let us evaluate the performance of our indexing schema. It is hard to compare our proposed schema to existing indexing techniques since we don't know of other indexing techniques except linear scanning which are able to meet our requirements.

Two of the main high-dimensionality indexing schemas are X-trees[2] and VA-file[12]. The X-Tree method is outperformed by a sequential scan for a dimension exceeding 6 (see [13]) and VA-files are only applicable to similarity-search queries. Thus, we only compare our performances with the performances of the sequential scan method.

5.1 Some approximations

Let us make some simplifications and approximations in order to create a model of the proposed indexing schema.

Type of data : We only consider one data type (integers). The cost of a comparison between two values is therefore always the same. This is not the case in real life, where data typically consist of numbers, booleans and strings. However, it is always possible to express the comparison cost of a data item type as a factor of a single integer comparison.

Optimizations : No optimization of the query processing on tag collections are taken into account. This means that tag collections are searched sequentially. Thus, the gain

obtained by querying tag collections is really the minimum we can expect from the new schema.

Data transfers : No optimization of data transfers are taken into account. Especially, we do not consider pipelined schema where the data transfer of a given item could be realized during the computation of the previous one.

Size of tag collections : For the performance analysis we consider only a single tag collection with a fixed number of tags. The number of tags and the size of the tags may be considered as an average among the different values of a real life example.

Complex queries : We do not take into account complex queries that could only be processed by a dedicated program. In other words, step 4 of the selection process (Figure 2) does not occur here.

5.2 Theoretical model

Let us adopt the following notations :

- N is the number of items in the whole database;
- n is the average number of items in a given tag collection;
- D is the number of values in a data item i.e. its dimension;
- d is the number of values in a tag i.e. the dimension of the tag; we assume that all these values are tested;
- d' is the number of values that are not contained in the tag but still need to be tested (step 8 in Figure 2);
- T_{lat} is the latency of the network which is used to transfer the data;
- T_{tr} is the time used to transfer one value through the network; in second per value;
- T_{IO} is the time needed to load one value from disk into the memory;
- T_{CPU} is the time to compute one value, i.e. to compare it with another value;
- q is the number of matching tags for the query we are dealing with;
- t_{seq} is the duration of the query using a sequential scan;
- t_{tag} is the duration of the query using the new indexing schema.

In the case of a sequential scan, the time needed to process a query is simply the time needed for querying one data item multiplied by N. Each data item is read from disk, transfered through the network and processed.

$$t_{seq} = N\left(T_{lat} + D\left(T_{tr} + T_{IO}\right) + \left(d + d'\right) T_{CPU}\right)$$

It is independent of the size of the result.

The time needed to process a query using the new indexing schema is slightly more complicated to compute. Using the architecture depicted in Figure 2, we can divide it into two parts : the duration t_1 of the query on tags and the duration t_2 of the query on data items. The query on tags is carried out on the server. Matching tags are transfered to the client. The query on data items is similar to the sequential scan method.

$$t_{tag} = t_1 + t_2$$

$$t_1 = n\,(d\,T_{IO} + d\,T_{CPU}) + q\,(T_{lat} + d\,T_{tr})$$
$$t_2 = q\,(T_{lat} + D\,(T_{IO} + T_{tr}) + d'\,T_{CPU})$$

Finally :

$$t_{seq} = N\,T_{lat} + N\,D\,(T_{IO} + T_{tr}) + N\,(d + d')\,T_{CPU} \tag{1}$$

$$t_{tag} = 2\,q\,T_{lat} + (n\,d + q\,D)\,T_{IO} + q\,(d + D)\,T_{tr} + (n\,d + q\,d')\,T_{CPU} \tag{2}$$

The query duration is dependent on the number q of matching tags. Note that the assumption that tags are transfered one by one to the client corresponds to the worst case. This could be improved by sending tags by groups.

5.3 Interpretation

The terms in equations (1) and (2) can be divided into three parts : processing time (T_{CPU}), network transfer time (T_{lat} and T_{tr}) and data retrieval time (T_{IO}). Let us consider them separately here.

Processing time : the processing time ratio between tag collection access and the default sequential scan is :

$$r_{CPU} = \frac{n\,d + q\,d'}{N\,(d + d')} = \alpha\,\frac{d + \gamma d'}{d + d'} \tag{3}$$
$$\textit{where} \qquad \alpha = \frac{n}{N} \qquad \gamma = \frac{q}{n}$$

Since $\gamma \leq 1$ (comes from $q \leq n$), we can be sure that $r_{CPU} \leq \alpha$. This demonstrates that the CPU time ratio is less than (but of the order of) the ratio between the number of tags in a collection and the number of data items.

Network transfer time : the network transfer time ratio between tag collection access and the default sequential scan is :

$$\begin{aligned} r_{NET} &= \frac{2\,q\,T_{lat} + q\,(d + D)\,T_{tr}}{N\,T_{lat} + N\,D\,T_{tr}} \\ &= \frac{q}{N}\,\frac{2\,T_{lat} + (d + D)\,T_{tr}}{T_{lat} + D\,T_{tr}} \end{aligned}$$

Since $d \leq D$, we finally have :

$$\begin{aligned} r_{NET} &\leq 2\,\frac{q}{N} \\ r_{NET} &\leq 2\,\alpha\,\gamma \end{aligned} \tag{4}$$

$$where \qquad \alpha = \frac{n}{N} \qquad \gamma = \frac{q}{n}$$

Since $\gamma \leq 1$, the network transfer ratio is at least of the order of the ratio between the number of tags in a collection and the number of data items. In practice, we even have $\gamma \ll 1$ (we foresee $\gamma \sim 10^{-2}$ for LHCb) and thus $r_{idle} \ll \alpha$.

Data retrieval time : the data retrieval time ratio between tag collection access and the default sequential scan is :

$$r_{DR} = \frac{n\,d + q\,D}{N\,D} = \alpha\,(\beta + \gamma) \tag{5}$$

$$where \qquad \alpha = \frac{n}{N} \qquad \beta = \frac{d}{D} \qquad and \qquad \gamma = \frac{q}{n}$$

Usually, $\beta \ll 1$ and $\gamma \ll 1$. Thus $r_{DR} \ll \alpha$. This means that, in respect to data retrieval time, we gain far more than just the gain obtained by the preselection on data items.

Let us estimate γ. By definition, γ is the proportion of matching tags in a tag collection for a given query. Let us consider a very simple case where every part of the query is a comparison and is fulfilled by half of the items. In addition, let us suppose that the data is uniformly distributed. This leads to :

$$\gamma = \frac{1}{2^d} \qquad and \qquad \frac{r_{DR}}{\alpha} = \frac{d}{D} + \frac{1}{2^d}$$

Figure 3 gives the behavior of this ratio against the dimension d for different values of D.

Roughly, $\frac{r_{DR}}{\alpha}$ goes down from 1 to a minimum for dimensions between 0 and $d_m \sim 8$ and linearly goes up afterwards until it reaches 1 again for dimension D. Clearly, we can approximate r_{DR} by $\alpha\frac{d}{D}$ if $d \geq d_m$. This is exactly our goal since the data retrieval time becomes proportional to the loading time of the tags.

For the LHCb experiment, the dimension of a data item is $D \sim 20000$. The minimum I/O time is reached for $d \sim 18$ and $r_{DR} < \frac{\alpha}{1000}$.

6 Conclusions

We presented a new way of indexing and selecting data in huge datasets having a high index dimensionality. The method avoids linear scanning of the whole data set. Only a minimal set of data is scanned, namely the values stored in tag collections. The selected tags point to the data items that are then retrieved for applying a more narrow selection.

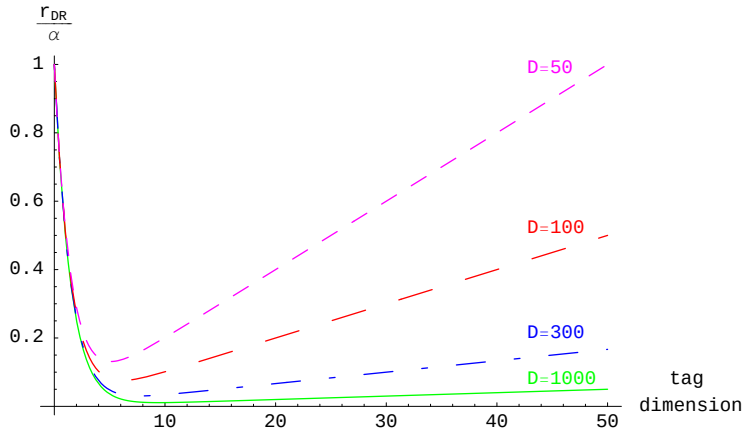


Figure 3: Evolution of a majoration of the data retrieval ratio divided by α in function of the dimension of the tag

By scanning tags in tag collections instead of a flat scan of all data items, the minimal gain is proportional to the ratio between the number of data items and the number of tags within the selected tag collections. In many cases, the effective gain is the minimal gain multiplied by the ratio of the dimension of data items and the dimension of tags.

The proposed data items selection and retrieval schema was implemented at CERN, in the context of the LHCb experiment and seems very promising. No enhancements have been tested at this time but an implementation of a computer assisted parallelization is planned.

References

[1] C. C. Aggarwal and P. S. Yu. The IGrid index: reversing the dimensionality curse for similarity indexing in high dimensional space. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 119–129, August 2000.

[2] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: an index structure for high-dimensional data. In *VLDB'96, Proc. of 22th International Conference on Very Large Data Bases*, pages 28–39, 1996.

[3] A. Guttman. R-trees : A dynamic indexing structure for spatial searching. In *SIGMOD'84*, pages 47–57, 1984.

[4] H. V. Jagadish, N. Koudas, and D. Srivastava. On effective multi-dimensional indexing for strings. In *Proc. 2000 ACM SIGMOD on Management of data*, pages 403–414, May 2000.

[5] N. Katayama and S. Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 369–380, May 1997.

[6] B. S. N. Beckmann, H-P. Kriegel R. Schneider. The R*-tree : An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD'90*, pages 322–331, 1990.

[7] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The subspace coding method: a new indexing scheme for high-dimensional data. In *Proceedings of the ninth international conference on Information knowledge management CIKM 2000*, November 2000.

[8] K.-T. Song, H.-J. Nam, and J.-W. Chang. A cell-based index structure for similarity search in high-dimensional feature spaces. In *Proceedings of the 16th ACM SAC2001 symposium on on Applied computing*, pages 264–268, March 2001.

[9] C. web pages. A matter of symmetry. URL : http://lhcb-public.web.cern.ch/lhcb-public/html/symmetry.htm.

[10] C. web pages. Experiments in B-physics. URL : http://lhcb-public.web.cern.ch/lhcb-public/html/bphysicsexpts.htm.

[11] C. web pages. What is CP-violation? URL : http://lhcb-public.web.cern.ch/lhcb-public/html/introduction.htm.

[12] R. Weber and S. Blott. An approximation-based data structure for similarity search. Technical Report 24, ESPRIT project HERMES (no. 9141), October 1997. Available at http://www-dbs.ethz.ch/∼weber/paper/HTR24.ps.

[13] R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proc. of 24th International Conference on Very Large Data Bases*, pages 194–205. Morgan Kaufmann, 1998.

Data Placement for Tertiary Storage

Jiangtao Li
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 U.S.A.
jtli@cs.purdue.edu
Phone: + 1 765 494-6008
Fax: + 1 765 494-0739

Sunil Prabhakar
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 U.S.A.
sunil@cs.purdue.edu
Phone: + 1 765 494-6008
Fax: + 1 765 494-0739

1 Abstract

In this paper we address the important problem of data placement in tertiary storage taking object relationships into account. This work is in contrast to earlier schemes that either focus on specific data types or assume that data objects are accessed independently. Five new data placement schemes are developed. The effectiveness of these schemes is shown through simulation. The proposed schemes, in particular the Edge Merge scheme, give superior performance over schemes optimized for independent access.

We also show that our schemes can easily adapt to variations in the access pattern. This also allows the schemes to be employed when no prior information about the access pattern is available. Interestingly, our results show that the probabilities of object access do not have a big impact on performance. On the other hand, changes to the clustering of nodes have a significant effect. This result underscores the importance of the relationships between objects for placement of data. The use of controlled replication for "free" is also developed and shown to be effective in further improving performance. The study also evaluates the impact of a secondary disk layer and prefetching.

2 Introduction

The tertiary storage layer in a hierarchical storage system is characterized by very large data volume and very high random access latency. Both attributes are directly related to the use of numerous cheap removable media sharing a small number of expensive drives and robot mechanisms. The high access latency is typically dominated by media switch time (for certain tape systems, however, the seek time may exceed the media switch time). With ever increasing demands for storing very large volumes of data for applications such as telemedicine, online multimedia document systems, and other large multimedia repositories, large amounts of live data are being stored on tertiary storage systems. Random

accesses to data stored on tertiary storage can suffer unacceptable delays as media are swapped on drives. The need for swapping media is dictated by the placement of data. Judicious placement of data on tertiary storage media is therefore critical, and can significantly affect the overall performance of the storage system.

The placement of data for specific domains such as multi-dimensional arrays [1], relational databases [15], and satellite images [21] has been addressed earlier. Research on tertiary storage placement in a more general setting has been addressed under the assumption that the data objects are accessed independently [2]. This assumption is rarely valid in practice – data objects typically are related and this is reflected in the access of the data. For example, online manuals contain hyperlinks to related sections and other manuals, a browsing session in a multimedia repository is typically guided by similarity between objects, and various test results of a given patient are likely to be accessed during diagnosis or treatment. In this paper we address the problem of placement of data on tertiary storage in a general setting without the assumption of independent access. Our approach is to exploit the nature of the access to the data to determine an optimal placement. This work is orthogonal to related issues of data migration and scheduling. The problem of placement of data on tertiary storage can be broken down into two sub-problems due to the significant cost of switching media: i) allocation of data to media; and ii) placement of data within the assigned medium. The problem of placing data within media has received some attention and we employ existing solutions to this problem such as [2]. The focus of our study is on the sub-problem of allocating data to media in order to minimize switching.

We propose and evaluate several placement schemes for tertiary storage systems based upon data access patterns. The schemes can be employed even if the access pattern is not known *a priori*, and can dynamically adapt to changes in access patterns. The study considers the impact of the secondary storage buffer and caching policy on the placement, and effective use of prefetching based upon the placement and access pattern. In an earlier study we demonstrated that for the case of multimedia documents replication of objects is an effective technique for reducing switching and improving performance. We study the use of replication of objects on multiple media for the general case in this study. The effectiveness of the proposed schemes is evaluated using a detailed hierarchical storage simulator. Our results show that significant improvements (as much as 80% reduction in average waiting time) can be achieved with our placement schemes. The remainder of the paper discusses the issues involved, our proposed approaches, and sample experimental results. Further details and results will be given in the full version of the paper.

3 Related Work

The placement of data for specific domains such as multi-dimensional arrays [1], relational databases [15], and satellite images [21] has been addressed earlier. Research on tertiary storage placement in a more general setting has been addressed under the assumption that the data objects are accessed independently. Placement schemes based upon independent

document access probabilities and no replication have been proposed in [2, 18]. Optimal arrangement of cartridges and file-partitioning schemes for carousel-type systems are investigated in [17]. Placement schemes for data on optical disks are developed in [3]. To the best of our knowledge, our work is the first to address the issues of placement of related objects (in a general setting) and replication.

Other researchers have addressed the use of hierarchical storage systems for multimedia data. A cache replacement technique for managing secondary storage buffers when multimedia objects are stored on tertiary storage has been developed by Ghandeharizadeh et al [6]. The use of a pipelining mechanism that avoids the need for complete materialization of an object on disk before initializing playback is presented in [5]. We have developed a prefix-caching scheme with low jitter and startup latency for storing continuous media data [14]. Storing video on hierarchical storage has also been studied in [20, 19]. The study addresses I/O bandwidth issues at the various levels of the storage hierarchy. Scheduling schemes for tertiary storage libraries are discussed in [4, 13, 8, 11] – any of these techniques can be applied in conjunction with our research to further improve performance. In [10] a prefetching algorithm based upon Markov-chain prediction of access is developed. Models of tape systems and tertiary storage system parameters can be found in [7, 9].

4 Data Placement Schemes

In this section we first explain the nature of access for related objects. This is followed by a description of the proposed tertiary placement schemes that take into account the relationships between objects. Then we discuss the issues of adaptive placement, impact of secondary storage, replication and prefetching.

4.1 Access Pattern for Related Objects

For efficient storage and retrieval of data it is critical to take into account the data access pattern. Data objects can be accessed either directly, or through a link from another object. Independent, or direct access to an object can be captured simply by the probability of access. In addition to direct access to objects, users may access objects based upon links from other objects (e.g HTML pages with links to other pages, or hyperlinks between manual pages). Such access is also very common in a browsing scenario whereby users simply follow links of interest. A user would typically begin by accessing an object and then possibly following some number of interesting links. If none of the links are interesting, the user may directly access some other object.

A *Browsing Graph* (BG) can be used to capture such access patterns. The browsing graph consists of labeled nodes and labeled edges. Each node represents an object and the label of the node gives the probability that the node will be accessed independently of the previous visited node. A directed edge between two nodes represents a link from one object to the other and the edge label gives the probability that the edge would be followed.

The sum of the probability of all edges going out of an object is not necessarily 1.0, since it is possible that none of the edges will be followed. We use the term *birth probability* to represent the probability of independent access to objects and *death probability* to represent the probability that once the node is accessed, none of its edges will be followed. The death probability of a node is simply 1 - (sum of outgoing edge probabilities).

4.2 Data Placement Schemes

Tertiary storage suffer from high access latency. The access cost in tertiary storage is dominated by the media exchange operation and head position delay. The goal of data placement is to minimize the expected access cost and reduce latency. In [2] it is shown that a placement whereby the objects are placed sequentially in decreasing order of their access probabilities is optimal. We call this the **Birth Probability Scheme**. This result, however, is based upon the assumption that the objects are accessed independently.

Static Probability Scheme: The frequency of an object being accessed is usually different from its birth probability. The object birth probability is the probability of the object being accessed directly, while the static probability is the probability of being accessed directly or indirectly. In other words, static probability represents the frequency of the object being requested. Given the user browsing graph, the static probability of an object can be easily computed by simulation. Our static probability data placement scheme is that the objects are placed sequentially in decreasing order of their static probabilities.

Edge Merge Scheme: This scheme explicitly takes into account the links between objects. Once an object is requested, it is very likely that objects with high probability links from this object will be accessed next. If such neighbors are placed on the same medium, a medium exchange can be avoided. The main idea of this scheme is therefore to place strongly related objects on the same medium. Ideally, all related objects are placed on the same medium. However, the medium capacity will not allow this. Therefore related objects may have to be spread across multiple media if the "cluster" of related objects is large. On the other hand, if there are small "clusters" then the problem is to pack as many clusters as possible on a single medium.

The basic idea behind edge merge is the following: Not all linked objects can be placed together; therefore, we give priority to higher probability links. To achieve this, we start merging objects that are linked by high probability edges into a new object. We define the new object's birth probability to be equal to the sum of that of the merged objects. Links into and out of the merged objects connect to the new object. Objects are merged in decreasing order of the link probabilities. Merging is not done if the the cumulative size of the resultant object will be larger than the medium capacity. When no further objects can be merged, the cumulative objects are allocated to media. This allocation follows the optimal scheme of [2] in decreasing order of the cumulative static probability.

Note that when two objects are merged, the cumulative birth probability is simply the

sum of the birth probabilities of the objects. Similarly, the probability for incoming edges from the same object are merged. For outgoing edges, a weighted sum of the probabilities is used if both merging objects have edges to the same object. The summing is done according to the static probability of the merging objects. The resulting static probability of the merged objects are computed in a manner similar to that explained earlier for the Static Probability scheme.

Hot Edge Merge Scheme: This scheme is very similar to the Edge Merge scheme. The only difference is that only edges that have a probability greater than a preset value (i.e. the "hot" edges) are merged. The idea is that this scheme will result in media with very high probability of access which will remain loaded most of the time.

Birth Hop Scheme: This scheme presents an alternative technique for combining direct and indirect access patterns. As in the hot edge merge scheme, we hope to use both object access probability and browsing graph information. The birth hop scheme works as follows. We begin by assigning the object with the highest birth probability to a blank medium. Following this step, we place as many objects as possible onto the same medium in decreasing order of either edge probability (from objects already allocated to the medium) or birth probability. Once the medium is full, we assign the object, from those that are unallocated, with the highest birth probability to a new medium and repeat the process. This operation is repeated until all objects are allocated.

Static Hop Scheme: This scheme is similar to birth hop scheme, except static probability instead of birth probability. The idea of this scheme is to allocate an object to a medium, we can either choose an object with highest static probability, or we can choose an object that has high probability edges with objects already on that medium.

4.3 Adaptive Placement

A key component of the proposed data placement schemes is knowledge of the access pattern. Although it is useful to know this a priori, it is not critical to the success of the proposed approach. Such information can easily be gathered from the system by keeping track of object requests. Based upon the observed access pattern, the data placement on tertiary storage can be tuned. In Section 5 we show the effectiveness of this adaptive placement in response to changes in the access pattern. In the complete absence of access information, the placement can begin with an initial guess for the access patterns followed by progressive refinement as user requests are serviced and the actual pattern is discovered.

4.4 Impact of Secondary Storage

In hierarchical storage systems, the secondary storage disks typically serve as a cache for data on tertiary storage. Depending upon the size of the disk layer and the caching (or migration) policy, some of the requests for objects are serviced directly from disk without impacting tertiary storage. The effect of the disk cache can be translated into a change in

the effective access pattern observed at the tertiary level. An adaptive strategy for tertiary storage can exploit this change in access pattern to generate a placement better suited for the available secondary storage cache.

4.5 Replication

Data objects that have strong links to objects in different media are likely to cause excessive swapping of media. While such situations will hopefully not arise often, it is possible that an object may have strong links to objects in different clusters. These two clusters may be placed on separate media due to their size. To overcome this, we propose to selectively replicate objects on multiple media based upon their edge probabilities to objects in various media. Furthermore, for schemes that place related collections of objects, it is possible that there are segments of media that not filled - these can be used to replicate objects for "free" since the extra space is not large enough for a cluster and would otherwise be empty.

4.6 Prefetching

Schemes that place collections of related objects together aim to avoid swapping of media for a sequence of requests from a user. It is possible, however, that in order to service the requests of other users, the media may be swapped. This could result in thrashing between the users and expensive swapping. To avoid this situation we investigate the use of prefetching of related objects from a medium before ejecting a loaded medium. Prefetching further delays pending requests and also uses up disk space. It is therefore important to make a good judgment about when and how much to prefetch.

5 Experimental Results

In this section we demonstrate the effectiveness of our new data placement schemes towards reducing average response time. The results are based upon a detailed CSIM [16] simulation model of the system. The tape library is modeled on the Ampex DST tape library configured with Ampex DST 310 drives [9]. Further details of the tape simulator can be found in [12]. The Secondary storage is configured with four 5GB disks, totaling 20 GB of disk storage. The tertiary storage component is modeled on a robotic tape library with four Ampex DST drives. Some of the important parameters for the tape simulation are provided in Table 1. The experiments were conducted on a synthetic collection of 10,000 objects of size 100 Megabytes each. The tape library is configured with 2000 tapes each of size 2GB, giving a total of 4TB of tertiary storage.

The set of objects and the access pattern is generated as follows. The birth probability of objects follows a Zipf distribution. In order to capture the effects of links between objects, we introduce the notion of edges between objects. To determine the edges, the objects are divided into clusters. The number of objects in a cluster is uniformly distributed between 5 and 20. Some (5%) of the objects are considered to be outliers that do not belong to any cluster. For each object, a *death probability*, p_d, is picked uniformly distributed between

Parameter	*Value(s)*	*Meaning*
TAPE SIMULATION PARAMETERS		
RWD_OVHD	0.0006 seconds	Rewind Overhead
SEEK_OVHD	0.0006 seconds	Seconds
SEEK_SPEED	110 MB/s	Tape seek rate
EJECT_TIME	4 seconds	Time to eject a tape
LOAD_TIME	10.1 seconds	Time to load a tape on a drive
PICK_TIME	3.7 second	Time for robot to grab a tape
PUT_TIME	1 second	Time for robot to drop a tape
MOVE_TIME	1.9 second	Time for robot to move
XFER_SPEED	14.2 MB/s	Tape transfer speed
NUM_TAPES	2000	Total number of tapes
TAPE_CAP	2 GB	Tape cartridge capacity
NUM_DRIVES	4	Number of Drives
DISK SIMULATION PARAMETERS		
ROT_SPEED	4002	Rotational speed RPM
SEC_TR	72	No. of sectors per track
CYLINDERS	1962	No. of cylinders
TR_CYL	19	No. of tracks per cylinder
TRKSKEW	8	Track skew in sectors
CYSKEW	18	Cylinder skew in sectors
CNTRL_TIME	1.2	Controller overhead (ms)
CAPACITY	5 GB	Disk storage capacity

Table 1: Table of Parameters

0.05 and 0.2. This is the probability that the user does not follow any of the links from this object. Edges to other objects within the cluster are created and assigned probabilities that are uniformly distributed so as to add to 1 - p_d.

It is important to note that although the access pattern is an input to the placement algorithm, it is not crucial that this pattern be accurate. As mentioned earlier, if the access pattern is unknown or changes after the placement, the system can adapt by reorganizing the data according to the new observed access pattern. Experimental evidence to support this claim is presented in Subsection 5.2.

In each experiment, we run a stream of requests. The stream begin by requesting a starting object identified using the birth probability for that object. As soon as this object is retrieved, the user chooses to either follow one of the edges from this object, or to pick another object independently. This choice is based upon the edge probabilities and the death probability of the currently accessed object. In each test, we run 1000 requests based upon which we compute the average response time.

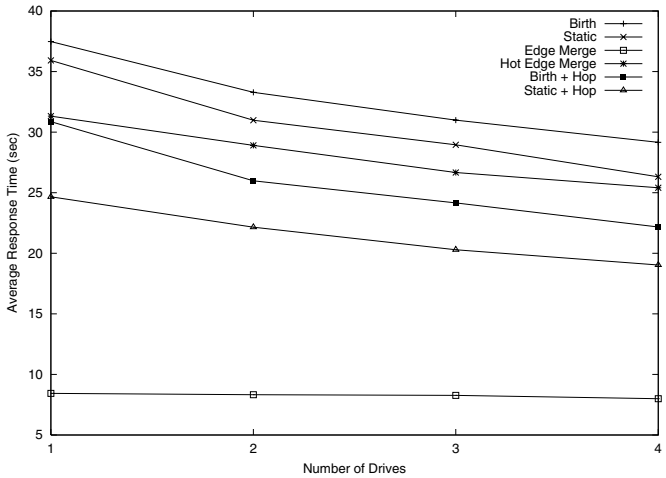


Figure 1: Average Response Time for Different Data Placement Schemes

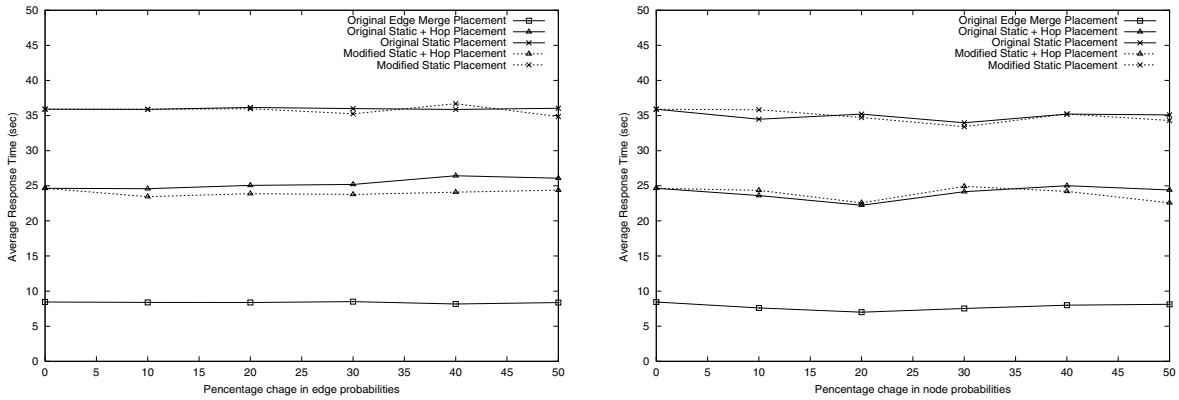
5.1 Different Data Placement Schemes and Performance

We begin by studying the relative behavior of the different schemes in reducing average response time. Figure 1 shows the average response time by different schemes. The number of drives was varied from 1 to 4. As can be seen from the graph, the $Edge\ Merge$ scheme gives the best performance, and the $Birth$ scheme has the worst performance. The $Static$ scheme has less average response time than $Birth$ scheme. The Edge Merge scheme reduces the average access time by 77% compared to the Static scheme for a single drive. We can also observe that as the number of drives increases, the average response time reduces for all schemes. The superior performance of *Edge Merge* was observed in all our experiments. The scheme that does not consider the relationships between objects (Birth) has the poorest performance. Similarly, the Static scheme has poor performance since it does not use the link information effectively.

5.2 Adapting to Variations in Access Pattern

In the preceding experiment it was assumed that the access pattern is known a priori. This information is used to generate the placements. If the access pattern is unknown or changes after the placement, the placement may be less beneficial. The actual access pattern can easily be discovered by recording the requests for objects. Based upon this input, a more effective placement can be achieved. Note that through observation, it is not possible to distinguish between direct and indirect access to an object. When object j is requested following a request for object i, it is not clear whether or not j was accessed due to a link from i to j. Consequently, the schemes based upon birth probability would not be applicable. We now investigate the impact of these variations.

In Figures 2 (a) and (b) we study the impact of random changes in the object access probabilities and the edge probabilities respectively. In each experiment the placement is



(a) (b)

Figure 2: Impact of Changes in (a) Edge; and (b) Node probabilities

generated based upon an initial access pattern. Next, a random subset of 10% of the nodes (edges) are chosen and their probabilities are altered to varying degrees. The performance is tested using this altered access pattern. The frequency of access to documents based upon this altered graph is captured and a new placement is made based only upon these observed frequencies (with no other knowledge of the changed access pattern). Using this adapted placement, the performance is again measured. This is repeated for varying degrees of changes from the original access pattern. From the graphs we observe that changes in edge and node probabilities have very little impact on the data placement schemes. These experiments show the impact of changes in the distribution of the node and edge probabilities while keeping the structure of the access pattern fixed. In other words, the results showed that if we know the groups of objects that are related, exact knowledge of the probabilities is not critical.

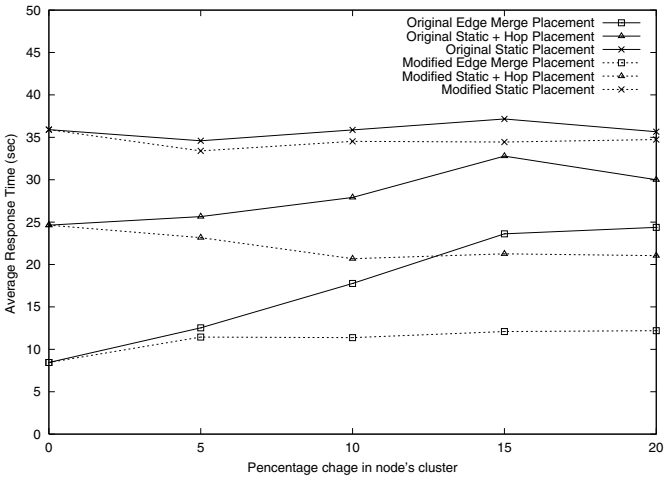


Figure 3: Impact of Changes in Node's cluster

In this experiment we study the impact of poor knowledge (or lack of knowledge) about

the grouping of related objects. In Figures 3 we study the impact of limited random changes in the object cluster composition. The placement is generated based upon an initial access pattern. Next, a random subset of 5%, 10%, etc of the nodes are chosen and the node's cluster membership is changed. The performance is tested using this altered access pattern. We also measure the performance of an adapted placement based upon the observed access pattern. As can be seen in the graph, changes to cluster composition result in an increase in the average response time for both placement schemes. However, we see that after adapting to the new pattern, we are able to reduce the response time. The response time is reduced sharply in $Edge\ Merge$ scheme, it drop to same level as no change to the access pattern. We can also notice that even without adapting to the new placement, the $Edge\ Merge$ scheme still performs better than $Static$ scheme.

From these three graphs we see an interesting result: information about the clustering or grouping of related objects is more critical than exact information about the probabilities of access. This is good news since these relationships are generally easy to discover statically based upon the application semantics (e.g. urls in a given web page). The results also underscore the importance of not making the assumption of independent access.

5.3 Impact of Secondary Storage

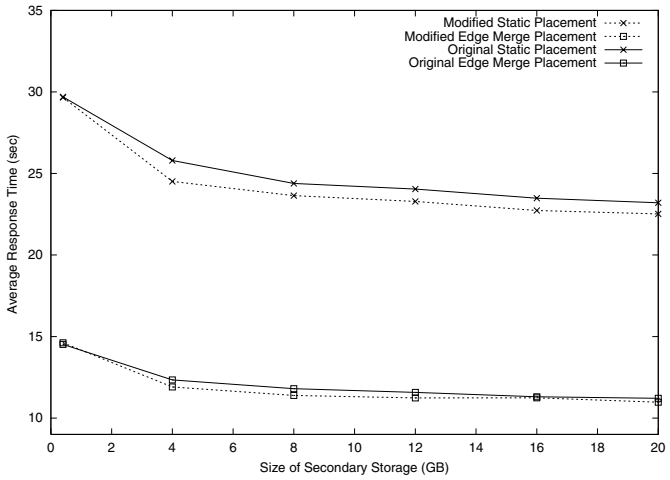


Figure 4: Impact of Secondary Storage

In this experiment, we study the impact of the size of the disk buffer. In hierarchical storage system, the secondary storage disks typically serves as a cache for data on tertiary storage. User requests for data cached in the buffer are served without any access to tertiary storage. If the requested object is not in the disk cache, the object is copied from tertiary storage to buffer, then from the buffer to the user. A buffer replacement policy is used to create space when the buffer becomes full. In our experiments, we use the popular Least Recently Used (LRU) cache replacement policy.

Figure 4 presents the performance for the various schemes for different buffer sizes. The buffer size is varied from 400MB upto 20GB. We can observe from the graph that as the size of the buffer increases, the average response time decreases for all schemes. We also observe that the presence of a disk cache does not change the relative performance of the $Edge\ Merge$ scheme and the $Static$ scheme.

Since we have secondary storage as cache. The effect of the disk cache can be translated into a change in the effective access pattern observed at the tertiary level. The hot objects (objects with high static probability) may not be hot at the tertiary level since these objects may always be cached on disk. In order to account for this change in the access pattern, we can adapt the placement based upon the observed access pattern at the tape level as was done in Section 5.2. In Figures 4, we study our new data placement based on observed access pattern. As we can seen from the graph, the new adapted data placement slightly better than original data placement.

5.4 Impact of Replication

In our original model, each object only has one copy in tertiary storage. To replicate objects on tertiary storage, there are two approaches. The first approach is to replicate some frequently requested objects. We can use this approach with the $Birth$ and $static$ schemes. However, disk caching will reduce the effectiveness of this approach because most of hot objects will reside in cache. The second approach is to replicate related objects when free space is available on a medium. This approach works for $Edge\ Merge$ scheme and $Hot\ Edge\ Merge$ scheme. In our experiment, we mainly study the $Edge\ Merge$ scheme with the second approach due to its superior performance. Unused segments on a medium are filled using the following rules. First objects that have strong connections with objects already in the medium are replicated. If space still remains after considering such objects, hot objects are replicated. The results of the experiment are shown in Figure 5. It can be seen that the free data replication results in a noticeable improvement in performance.

5.5 Prefetching Issues

As stated in the last section, prefetching related objects can be beneficial. The disadvantage is that prefetching delays pending requests further and uses up disk space. In this subsection, we study the impact of prefetching for the proposed schemes. In order to see the impact of the amount of prefetching performed, we tested our six schemes with different prefetching sizes. In this experiment prefetching is performed whenever possible. When a new tape is loaded onto the drive, any object not in the disk cache may be prefetched. The total amount of data prefetched from a single medium is varied from 0 to 300 MB. The results of the experiment are shown in Figure 6. As can be seen, most schemes benefit from prefetching when the prefetch size is 100 MB. For larger sizes, only the Edge Merge scheme benefits – the average response time is reduced by 13%. This is explained by the fact that the Edge Merge scheme is based on the relationship between objects. When one object is retrieved, the most connected objects are likely to be in the same medium, so

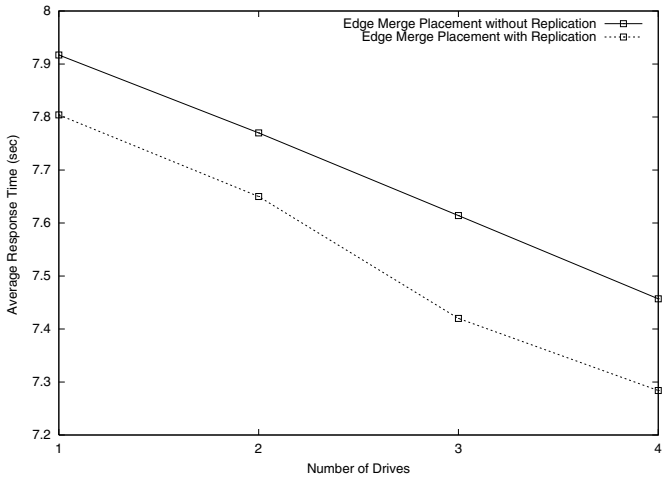


Figure 5: Impact of Replication on Edge Merge Scheme

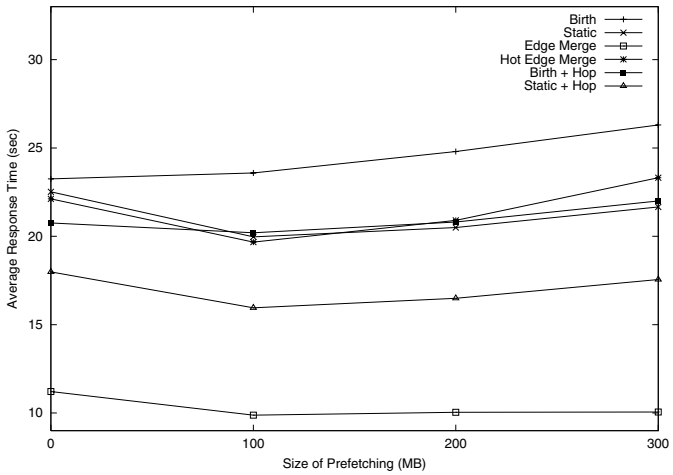


Figure 6: Impact of Prefetching on Different Schemes

prefetching is beneficial. Prefetching is not good for the Birth Scheme because under this scheme related objects are scattered in different media. In fact the penalty of prefetching larger than 100MB of data is higher than the benefit.

Next we study the choice of when to prefetch with the Edge Merge scheme in order to make prefetching most effective. In the last experiment we prefetched blindly. In this experiment, we prefetch only if there is a suitable object. There are two kinds of candidates for prefetching when a medium is loaded for retrieving object O_i: i) objects with strong links from O_i; and ii) objects with a large static probability. The experiment is controlled by two parameters: a minimum edge probability (say min_{ep}) and a minimum static probability (say min_{sp}). If an object in same medium has edge probability greater than min_{ep} or static probability greater than min_{sp}, that object will be a prefetching candidate. Since we cannot

prefetch all candidates the amount of data prefetched is limited. The results are shown in Figure 7. Only Edge Merge is studied, with several prefetching policies. We study 4 policies: i) min_{ep} = 0.5, min_{sp} = 0.005, this is most restrictive policy; ii) min_{ep} = 0.3, min_{sp} = 0.0003, preference is given to high static probability objects; iii) min_{ep} = 0.05, min_{sp} = 0.005, preference is for high edge probability objects; and iv) min_{ep} = 0.05, min_{sp} = 0.0003, this is the most liberal policy. As we can observed from graph the most liberal policy gives better performance than the most strict policy. The two schemes that have a low threshold for the edge probability give better performance for small prefetch sizes, but their performance degrades for larger prefetch sizes.

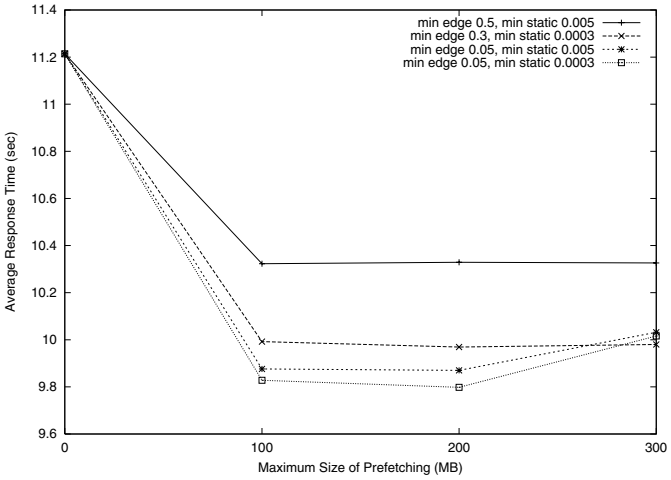


Figure 7: Impact of Different Prefetching Policies on Edge Merge Scheme

6 Conclusion

In this paper we address the important problem of data placement in tertiary storage taking object relationships into account. We also study the advantage of limited replication in this setting. This work is in contrast to earlier schemes that either focus on specific data types or assume that data objects are independently accessed. To the best of our knowledge, this is the first study to explore these issues. We propose five new data placement schemes. The effectiveness of these schemes in reducing average response time is shown through extensive experimentation using a detailed simulator. We find the Edge Merge scheme has best performance. The performance of placement schemes that are known to be optimal under the assumption of independent access is not as good as that of the proposed schemes.

We also show that our schemes can easily adapt to variations in the access pattern. In fact this allows the schemes to be employed when no prior information about the access pattern is available. The schemes progressively adapt to give good performance as the access pattern is learned. Capturing the access pattern is easily achieved at the tertiary storage level. In all cases, adjusting the placement to the new observed pattern resulted in

significantly improved performance. Interestingly, our results show that the probabilities of access (node and edge) do not have a big impact on our Edge Merge scheme. Changes to the clustering of nodes, on the other hand, has a greater effect. This goes to show the importance of the inter-relationships between objects. The use of controlled replication for "free" is also developed and shown to be effective in improving performance further. The impact of disk caching is easily handled in a manner similar to that of variation in access patterns. The effective access pattern at the tertiary layer is measured and used to place the data, rather than the overall access pattern. The techniques are coupled with prefetching which is found to be beneficial for the Edge Merge scheme.

Overall, we see that the proposed techniques are very effective in placing data on tertiary storage. The techniques perform much better than schemes that are optimal under the assumption of independent access. In our experiments the Edge Merge scheme achieved as much as 77% reduction in average access time over the state-of-the-art scheme (Static).

Acknowledgment This work was supported by the National Science Foundation under CAREER grant IIS-9985019, and Research Infrastructure Grant 9988339-CCR.

References

[1] L. T. Chen, R. Drach, M. Keating, S. Louise, D. Rotem, and A. Shoshani. Efficient organization and access of multi-dimensional datasets on tertiary storage systems. In *Information Systems*, volume 20, pages 155–83. Elsevier Science, 1995.

[2] S. Christodoulakis, P. Triantafillou, and F. Zioga. Principles of optimally placing data in tertiary storage libraries. In *VLDB'97, Proc. of Intl. Conf. on Very Large Data Bases, 1997, Athens, Greece*, pages 236–245, 1997.

[3] D. A. Ford and S. Christodoulakis. Optimizing random reterievals from clv format optical disks. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 413–22, Barcelona, Spain, September 1991.

[4] C. Georgiadis, P. Triantafillou, and C. Faloutsos. Scheduling and performance of robotic tape libraries in video server environments. Technical report, Multimedia Systems Institute of Crete (MUSIC), Tech. Univ. of Crete, Crete, Greece, 1997.

[5] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. Pipelining mechanism to minimize the latency time in hierarchical multimedia storage managers. *Computer Communications*, 18:170–184, march 1995.

[6] S. Ghandeharizadeh and C. Shahabi. On multimedia repositories, personal computers, and hierarchical storage systems. In *Proc. of ACM Int. Conf. on Multimedia*, 1994.

[7] B. K. Hillyer and A. Silberschatz. On the modeling and performance characteristics of a serpentine tape. In *SIGMETRICS*, pages 170–9, Canada, 1996.

[8] B. K. Hillyer and A. Silberschatz. Random I/O scheduling in online tertiary storage. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Canada, 1996.

[9] T. Johnson and E. L. Miller. Performance measurements of tertiary storage devices. In *Proc. of 24rd Intl. Conf. on Very Large Data Bases*, pages 50–61, New York, 1998.

[10] A. Kraiss and G. Weikum. Vertical data migration in large near-line document archives based on markov-chain predictions. In *Proceedings of 23rd International Conference on Very Large Data Bases*, pages 246–255, Athens, Greece, August 1997.

[11] S. More, S. Muthukrishnan, and E. Shriver. Efficiently sequencing tape resident jobs. In *Proc. ACM Symp. on Principles of Database Systems*, 1999.

[12] S. Prabhakar. An overview of current tertiary storage technology and research. Master's thesis, University of California, Santa Barbara, 1998.

[13] S. Prabhakar, D. Agrawal, A. El Abbadi, and A. Singh. Scheduling tertiary I/O in database applications. In *Proc. of the 8th International Workshop on Database and Expert Systems Applications*, pages 722–727, Toulouse, France, September 1997.

[14] S. Prabhakar and R. Chari. Minimizing latency and jitter for large scale multimedia repositories through prefix caching. Technical Report CSD 01-018, Department of Computer Sciences, Purdue Univeristy, September 2001.

[15] S. Sarawagi. Database systems for efficient access to tertiary memory. In *Proc. of 14th IEEE Symp. on Mass Storage Systems*, pages 120–6, Monterey, California, 1995.

[16] H. D. Schwetman. CSIM: A C-based, process-oriented simulation language. In *Proceedings of the 1986 Winter Simulation Conference*, pages 387–396, December 1986.

[17] S. Seshadri, D. Rotem, and A. Segev. Optimal arrangements of cartridges in carousel type mass storage systems. *The Computer Journal*, 37(10):873–887, 1994.

[18] P. Triantafillou, S. Christodoulakis, and C. Georgiadis. Optimal data placement on disks: A comprehensive solution for different technologies. Technical report, Multimedia Systems Institute of Crete (MUSIC), Tech. Univ. of Crete, Greece, 1996.

[19] P. Triantafillou and T. Papadakis. On-demand data elevation in hierarchical multimedia storage servers. In *Proc. of 23rd Intl. Conf. on Very Large Data Bases*, pages 226–235, Athens, Greece, August 1997.

[20] P. Triantafillou and T. Papadakis. Exploiting tertiary storage for performance improvement in video-on-demand servers. Technical report, Multimedia Systems Institute of Crete (MUSIC), Technical University of Crete, Crete, Greece, 1998.

[21] J. Yu and D. DeWitt. Processing satellite images on tertiary storage: A study of the impact of tile size on performance. In *5th NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 460–476, College Park, Maryland, Sept. 1996.

Storage Resource Managers: Middleware Components for Grid Storage

Arie Shoshani, Alex Sim, Junmin Gu
Lawrence Berkeley National Laboratory
Berkeley, California 94720
{shoshani, asim, jgu}@lbl.gov
tel: +1-510-486-5171
fax: +1-510-486-4004

Abstract

The amount of scientific data generated by simulations or collected from large scale experiments have reached levels that cannot be stored in the researcher's workstation or even in his/her local computer center. Such data are vital to large scientific collaborations dispersed over wide-area networks. In the past, the concept of a Grid infrastructure [1] mainly emphasized the *computational* aspect of supporting large distributed computational tasks, and managing the sharing of the *network* bandwidth by using bandwidth reservation techniques. In this paper we discuss the concept of Storage Resource Managers (SRMs) as components that complement this with the support for the *storage* management of large distributed datasets. The access to data is becoming the main bottleneck in such "data intensive" applications because the data cannot be replicated in all sites. SRMs are designed to dynamically optimize the use of storage resources to help unclog this bottleneck.

1. Introduction

The term "storage resource" refers to any storage system that can be shared by multiple clients. We use the term "client" here to refer to a user or a software program that runs on behalf of a user. Storage Resource Managers (SRMs) are middleware software modules whose purpose is to manage in a dynamic fashion what resides on the storage resource at any one time. SRMs do not perform file movement operations, but rather interact with operating systems, mass storage systems (MSSs) to perform file archiving and file staging, and invoke middleware components (such as GridFTP) to perform file transfer operations. There are several types of SRMs: Disk Resource Managers (DRMs), Tape Resource Managers (TRMs), and Hierarchical Resource Managers (HRMs). We explain each next. Unlike a storage system that allocates space to users in a static fashion (i.e. an administrator's interference is necessary to change the allocation), SRMs are designed to allocate and reuse space dynamically. This is essential for the dynamic nature of shared resources on a grid.

A Disk Resource Manager (DRM) manages dynamically a single shared disk cache. This disk cache can be a single disk, a collection of disks, or a RAID system. The disk cache is available to the client through the operating system that provides a file system view of

the disk cache, with the usual capability to create and delete directories/files, and to open, read, write, and close files. However, space is not pre-allocated to clients. Rather, the amount of space allocated to each client is managed dynamically by the DRM. The function of a DRM is to manage the disk cache using some client resource management policy that can be set by the administrator of the disk cache. The policy may restrict the number of simultaneous requests by each client, or may give preferential access to clients based on their assigned priority. In addition, a DRM may perform operations to get files from other SRMs on the grid. This capability will become clear later when we describe how DRMs are used in a data grid. Using a DRM by multiple clients can provide an added advantage of file sharing among the clients and repeated use of files. This is especially useful for scientific communities that are likely to have an overlapping file access patterns. One can use cache management policies that minimize repeated file transfers to the disk cache for remote grid sites. The cache management policies can be based on use history or anticipated requests.

A Tape Resource Manager (TRM) is a middleware layer that interfaces to systems that manage robotic tapes. The tapes are accessible to a client through fairly sophisticated Mass Storage Systems (MSSs) such as HPSS, Unitree, Enstore, etc. Such systems usually have a disk cache that is used to stage files temporarily before transferring them to clients. MSSs typically provide a client with a file system view and a directory structure, but do not allow dynamic open, read, write, and close of files. Instead they provide some way to transfer files to the client's space, using transfer protocols such as FTP, and various variants of FTP (e.g. Parallel FTP, called PFTP, in HPSS). The TRM's function is to accept requests for file transfers from clients, queue such requests in case the MSS is busy or temporarily down, and apply a policy on the use of the MSS resources. As in the case of a DRM, the policy may restrict the number of simultaneous transfer requests by each client, or may give preferential access to clients based on their assigned priority.

A Hierarchical Storage Manager (HRM) is a TRM that has a staging disk cache for its use. Thus, it can be viewed as a combination of a DRM and a TRM. It can use the disk cache for pre-staging files for clients, and for sharing files between clients. This functionality can be very useful in a data grid, since a request from a client may be for many files. Even if the client can only process one file at a time, the HRM can use its cache to pre-stage the next files. Furthermore, the transfer of large files on a shared wide area network may be sufficiently slow, that while a file is being transferred, another can be staged from tape. Because robotic tape systems are mechanical in nature, they have a latency of mounting a tape and seeking to the location of a file. Pre-staging can help mask this latency. Similar to the file sharing on a DRM, the staging disk in an HRM can be used for file sharing. The goal is to minimize staging files from the robotic tape system. The HRM design is based on experience in a previous project reported in [2].

The concept of an SRM can be generalized to the management of multiple storage resources at a site. In such cases, the site SRM may use "site-file-names" (directory path

+ file names) which do not reflect the physical location and file names. This gives the site the flexibility to move files around from one storage device to another without the site-file-names changing. When a client accesses a file using a site-file-name, it may be given in response the physical location and file name. The client can then use the physical file name to execute a file transfer.

In general, it is best if SRMs are shared by a community of users that are likely to access the same files. They can be designed to monitor file access history and maximize sharing of files by keeping the most popular files in the disk cache longer.

2. The role of SRMs in a Data Grid

Suppose that a client runs an analysis program at some site and wishes to get data stored in files located in various sites on the grid. First, the client must have some way of determining which files it needs to access. Checking a file catalog, using some index, or using a database system containing information about the files can accomplish this step. We refer to this step as "request interpretation". The information used in this step is often referred to as a "metadata catalog". The result of this step is a set of logical file names that need to be accessed. The second step is to find out for each logical file where it physically resides or replicated. Note that a single logical file can be replicated in multiple sites. Files can be either pre-replicated in multiple sites based on expected use by a system administrator or replicated dynamically because they were accessed by clients at these sites. In a grid environment, the information on the locations of replicated files exists in a "replica catalog", a catalog that maps a single logical file name to multiple site-specific files. The site-specific file name includes the name a machine and possibly port at the site, the directory path on that system, and the file name.

In many grid environments today, the burden for the above work is being thrust on the clients. Therefore, it is now recognized that such tasks can be delegated to middleware components to provide these services. A "request manager" is the term used to refer to such services. The request manager performs "request planning" based on some strategy, and then a "request execution" of the plan. This terminology is used by several grid projects, notably PPDG [3], GriPhyN [4], and ESG [5]. There are three options to consider for request planning: either move the client's program to the site that has the file, move the file to the client's site, or move both the program and the data to another site for processing. All three possibilities are valid, and much of the middleware development addresses this issue. In all these cases, SRMs play an important role. In the case that the program moves to the site where the file exists, it is necessary to "pin" the file in that site; that is, to request that the file remains in that site, so that when the program is executed the file is found in the cache. When the program completes, the file can be "released". In the case that the file needs to be transferred from a source site to target site (either to the client's site, or to another site), it is necessary to "pin" the file in the source site, to reserve the space in the target site, and maintain this state till the transfer to the target site is complete. Then the "pin" can be released. Here, the SRM at

the source site has the role of managing the "pinning", and the SRM at the target site has the role of allocating space (i.e. making space by removing other files if necessary), and reserving the space till the transfer completes. SRMs need to deal also with various failures, so that space reservations do not persist forever, and "pins" do not persist in case that a "release" is not performed. The concept of "pinning a file" is central to SRMs and will be discussed further later in this document.

In a recent paper [6], the authors describe 5 layers needed to support grid applications: fabric, connectivity, resource, collective, and application layers. The purpose of this layered approach is that services in each layer can rely on services in layers below it. The fabric layer consists of computational resources, storage resources, network resources, catalogs, code repositories, etc. The connectivity layer consists of communication, authentication, delegation, etc. The resource layer consists of components (and protocols) for managing various resources: computing, storage, network, catalog, inquiry, etc. We see SRMs as belonging to the "resource layer". The collective layer consists of services such as replica catalog, replica selection, request planning, and request execution. Request management is a generic term that uses any of the services in that layer, as well as services below it. The application layer consists of application specific services. The "request interpretation" we mentioned above belongs to this layer, since finding which logical files are needed by an application is specific to that application.

3. A practical use case: an analysis scenario

We describe below an analysis scenario where the computation is performed at the client's site, and the needed files are in other sites on the grid. This is a common special case of grid resource usage in many scientific communities. The schematic diagram of this analysis scenario is shown in Figure 1.

As shown in Figure 1, at the client's site there may be multiple clients sharing a local disk cache. Each of the clients issues a logical request, typically consisting of a logical predicate condition for what they wish to analyze. A typical example of such a request in the high-energy physics domain (where atomic particles are accelerated and made to collide at high speeds) might be: "find all the collisions (called "events") that have an energy more that 50 GEV, and produced at least 1000 particles". A similar request for climate model analysis may be "get all temperatures and wind velocity for summer months in the Pacific Ocean region for the last ten years". These requests may be produced by a graphical user interface or composed by the client using some query language. The Request Interpreter is a component that accepts the logical query and produces a set of logical file names that contain the desired data. A Request Planner may check with a Replica Catalog and other network services such as the "network weather service" (which provides an estimate of current network availability) to determine the replica site from which to get each file. The Request Executer then executes this plan. An example of a request executer, called DAGMAN (for Directed-Acyclic-Graph Manager) was recently developed by the Condor project [7].

The request executer could communicate with various SRMs on the grid, requesting space allocation and file pinning, and making requests for file transfers. However, we have decided to delegate the task of making requests for file transfers to the SRMs.

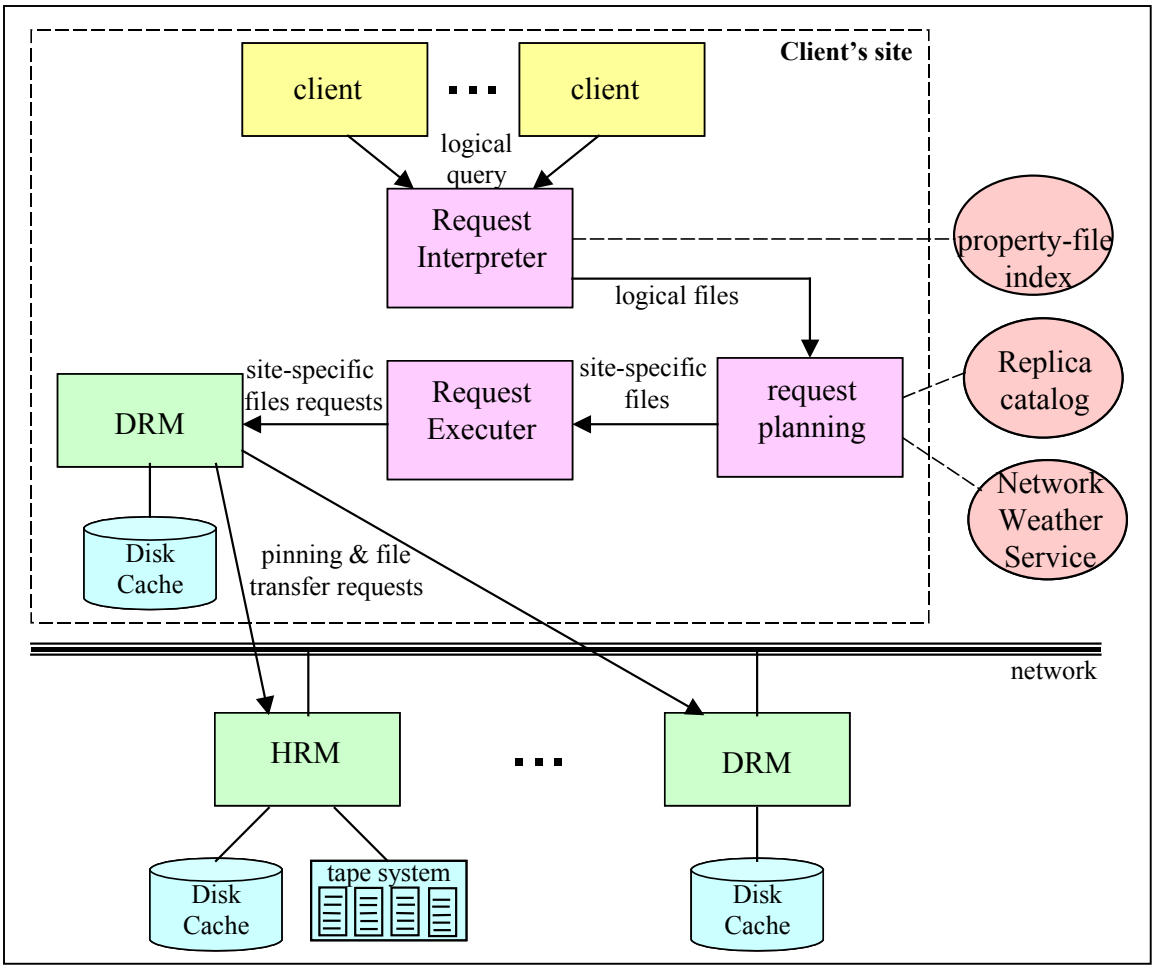


Figure 1. A schematic diagram of an analysis scenario

Specifically, if a request for a set of files is made to an SRM, it is its responsibility to dynamically allocate space for the files, to negotiate with remote SRMs the pinning of files at the remote site, to invoke file transfer services to get the files from other sites and to release the files after they are used. By making this fundamental design choice, we not only simplify the request executer's task, but also permit clients to communicate *directly* with SRMs making multi-file requests. The ability for clients to request files directly from an SRM was a basic requirement that guided our design since, in general, one cannot assume the existence of request managers. Furthermore, clients should be able to make direct requests to SRMs if they so choose. A secondary advantage of this design

choice is that it facilitates file sharing by the SRMs. Since clients can make multi-file requests to the SRM, it can choose to serve files to clients in the order that maximizes file sharing, thus minimizing repeated file transfers over the network.

For the analysis scenario shown in Figure 1, where all the files have to be brought to the local disk cache, the request executer makes its file requests to the local DRM. The local DRM checks if the file is already in its cache. If it is in the cache, it pins the file. If it is not, it communicates with other SRMs to get the files.

We have implemented several versions of DRMs as well as an HRM that interfaces to the HPSS mass storage system. The HRM is implemented as a combination of a TRM that deals with reading/writing files from/to HPSS, and a DRM for managing its disk cache. Both the DRM and the TRM are capable of queuing requests when the storage systems they interface to are busy. For example, a TRM interfacing with HPSS may be limited to perform only a few staging request concurrently, but it may be asked to stage hundreds of files. These requests are then queued, and performed as fast as HPSS will perform. The SRMs use grid-enabled secure file transfer services provided by the Globus project [8], called GridFTP. These DRM and HRM components are in the process of being used by one of the experiments of the Particle Physics Data Grid (PPDG) [3], and the Earth Science Grid (ESG) [5] to perform grid file replication functions. The HRM was also used in a demo for SuperComputing 2000 as part of an infrastructure to get files from multiple locations for an Earth Science Grid application (ESG). This was described in a recent paper [9]. We are now evaluating several "cache replacement policies" to be used by DRMs, by both conducting simulations and setting up real testbeds.

4. The implementation of the analysis scenario

The analysis scenario described in Figure 1 was implemented as part of a demo during the Supercomputing 2001 conference. The application used in the demo was high-energy physics (HEP). Figure 2 shows the actual setup of the demo. From a client's point of view the system accepts a logical query request, and takes care of all the details of figuring out what files should be transferred, and where to get them from. The client can observe in a graphical display the progress of file transfers over time. Figure 3 shows the progress of transfer of each file managed by the client's DRM. Partially filled bars represent transfer in progress. When a file that arrives is processed and released by the client, it may be removed automatically by the DRM if it needs to make space for additional files.

In order to illustrate the usefulness of SRMs, we describe next in some detail the steps of processing a logical query in a grid environment. In figure 2, the Bit-Map index is a specialized index used as the "request interpreter", which was developed as part of another project [10]. It gets as input a logical request made of logical conditions over range predicates. An example of such a request in this HEP application is to find all files that contain collisions (or "events") for which the following condition holds:

((0.1 < AVpT < 0.2) ^ (10 < Np < 20)) v (N > 6000),

where AvpT is the "average momentum", Np is "the number of pions" produced in this collision, and N is the "total number of particles produced in this collision". The result of the Bit-Map index is a set of logical file names, such as:

{star.simul.00.11.16.tracks.156,…, star.simul.00.11.16.tracks.978},

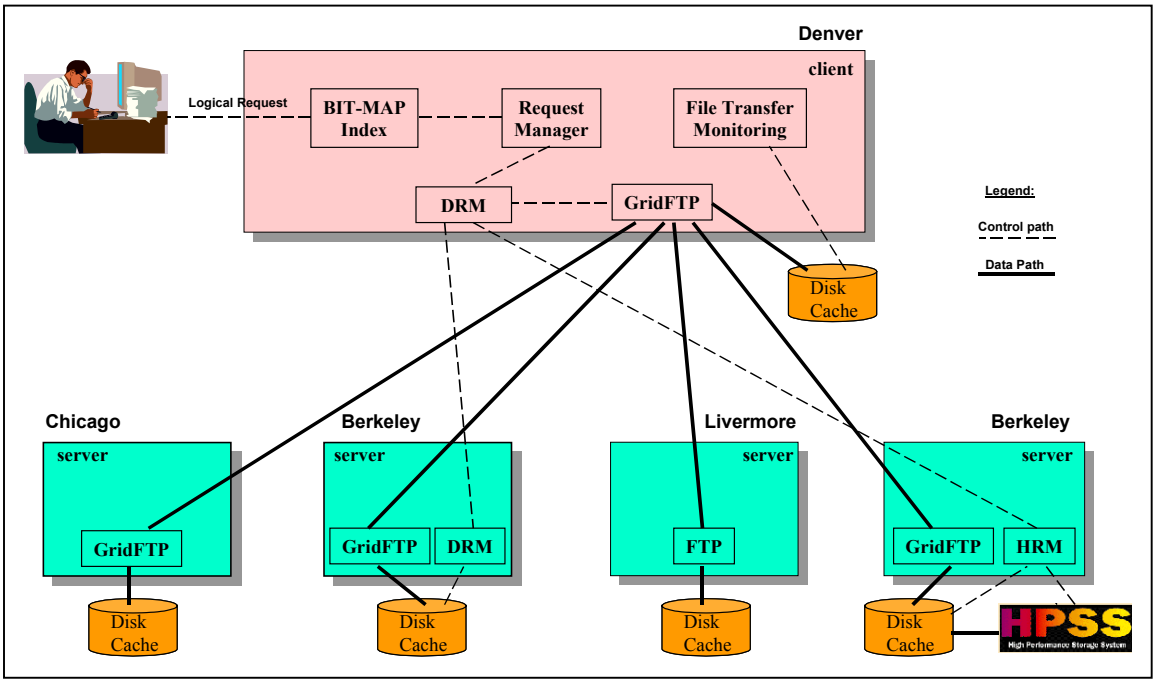


Figure 2. A setup for processing logical analysis requests over the grid

where "star" is the name of the experiment at Brookhaven National Laboratory, "simul" means simulation data, "00.11.16" is the date the data was generated, "tracks" refers to the type of data in the file, and the number is the file ID. This set of logical file names is given to the next component, the Request Manager.

The Request Manager (which consists of both a Request Planning and Request Execution components) is a component that chooses the site where to get each file, and then oversees the execution of the request. Given that a file may be replicated in multiple locations, it chooses the most appropriate location. Each file is assigned a "site file name" in the form of a URL, such as:

gsiftp://dg0n1.mcs.anl.gov/homes/sim/gsiftp/star.simul.00.11.16.tracks.156,

where "gsiftp" is the protocol for transferring the file, "dg0n1.mcs.anl.gov" is the machine name, "homes/sim/gsiftp" is the directory path, and

“star.simul.00.11.16.tracks.156” is the file name.

Similarly, if the site that has the file is managed by an SRM, the protocol used will say “hrm” or “drm”. For example, for accessing the same file out of an HPSS tape system, the URL used is:

hrm://dm.lbl.gov:4000/home/dm/srm/data1/star.simul.00.11.16.tracks.156,

where “dm.lbl.gov:4000” is the name of the machine that has HRM running on it, and the port used by HRM, “home/dm/srm/data1” is the directory on the HPSS system where the file resides, and “star.simul.00.11.16.tracks.156” is the file name.

Note that files can reside on systems that may or may not have an SRM managing the storage system. We set up the demo to illustrate that an SRM can work with systems managed by other SRMs, or systems that have some grid middleware (such as GridFTP), or even systems that have no middleware software at all (using only FTP to transfer files). In the demo, we set up four types of nodes: one with a DRM managing the storage system (at LBNL), one with an HRM managing access to an HPSS system (at NERSC-LBNL), one that has no SRM but has GridFTP available on it (at ANL), and one that has only FTP available on it (at LLNL).

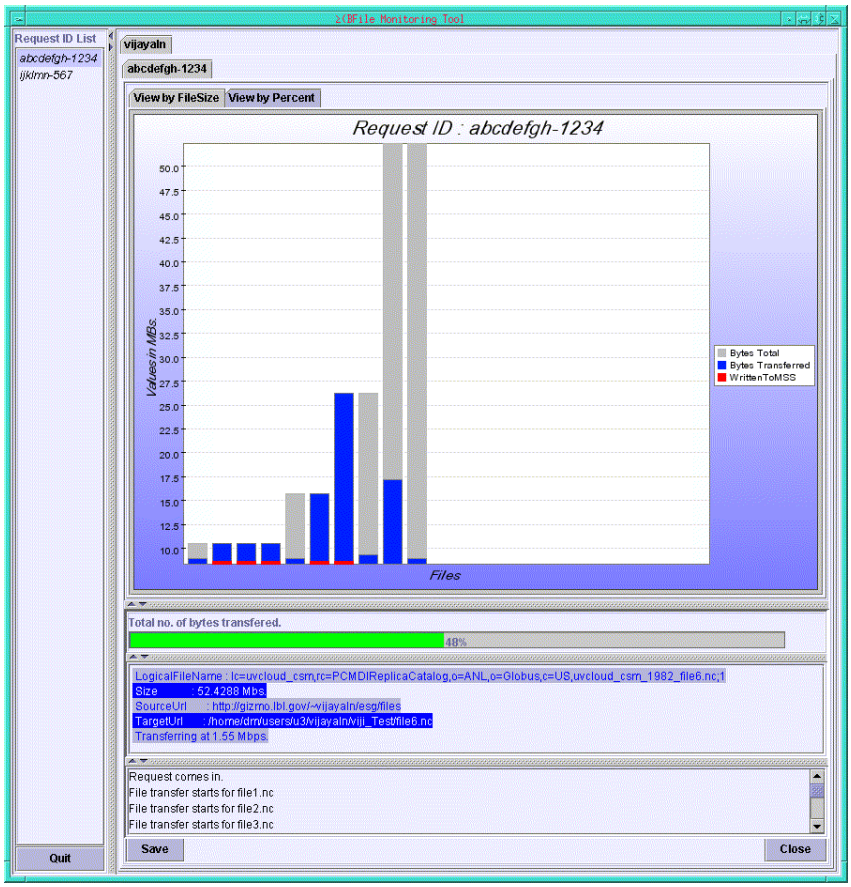


Figure 3. Display of the dynamic progress of file transfers

Once the Request Manager has assembled the set of URLs for the files needed, it invokes the local DRM (at the Supercomputing Conference floor at Denver). The local DRM then checks for each file if it is already in cache, and if the file is not found it contacts the site that has it, requesting pinning of files, and invoking the appropriate file transfer service (GridFTP or FTP in this demo). Once a file is transferred, it sends a "release of file" notice to the source site.

The SRMs are multi-threaded components that can support simultaneous file transfer requests from multiple clients. Thus, given a request for multiple files, the client's DRM will initiate the coordination of space reservation, pinning of files, and multiple file transfer requests to multiple sites. The number of such concurrent processing of file transfer requests is a policy decision. Since multiple clients may share a local DRM, the

DRM may have a policy to restrict the amount of space and the number of files that a client can hold simultaneously.

The display of file transfers in Figure 3 was designed to show dynamic progress. The local disk cache is checked every 10 seconds (a parameterized choice) for the size of files being transferred, and the display is updated. The horizontal bar below file progress display shows the total bytes transferred as a fraction of the total bytes requested. Moving the curser over any of the file bars provides information of the source location, size, and transfer rate. This is shown in the lower section of the display. Finally, there is a "message section" at the bottom to inform the client of events as they occur, including failures to access files and the reasons for that, such as "system down".

The above scenario was limited to cases where all the files are moved to the client's location. The generalization of this scenario is that the request planner generates a plan where the execution of the analysis can be partitioned to run on multiple sites (perhaps the sites where the data reside to minimize file transfer traffic). In this general scenario, both data and programs can move to the locations best suited to execute a request in the most efficient manner possible. The general scenario also includes moving the results of computations to the client, as well as storing results in storage systems and archives on the grid. Thus, in general, SRMs can be invoked at multiple locations by a single client to satisfy the request plan.

5. Advantages of using SRMs

As can be deduced from the discussion above, the main advantage of an SRM is that it provides smooth synchronization between shared resources by pinning files, releasing files, and allocating space dynamically on an "as-needed" basis. A reasonable question is why use SRMs if it is possible to use GridFTP and FTP directly as was done in the above demo. We recall that SRMs perform two main functions: dynamic space allocation and dynamic file pinning. Indeed, if space is pre-allocated, and the files are "permanently" locked in the source site there is no need for SRMs. However, in a grid environment where resources need to be reused dynamically, SRMs are essential. SRMs perform the management of quotas, the queuing of requests when resources are tight or if the clients exceed their quota, the freeing of space of files allocated but not released by clients (similar to "garbage collection"), and providing the management of buffers for pre-staging from mass storage systems. Pre-staging and buffering are important because the network bandwidth available to a client may vary in an unpredictable fashion.

A second advantage of using SRMs is that they can eliminate unnecessary burden from the client. First, if the storage system is busy, SRMs can queue requests, rather than refuse a request. Instead of the client trying over and over again, till the request is accepted, an SRM can queue the request, and provide the client with a time estimate based on the length of the queue. This is especially useful when the latency is large such

as for reading a file from tape. If the wait is too long, the client can choose to access the file from another site, or wait for its turn. Similarly, a shared disk resource can be temporarily full, waiting for clients to finish processing files, and therefore queuing requests is a better alternative than simply refusing the request.

A third advantage to the clients is that SRMs can insulate them from storage systems failures. This is an important capability that is especially useful for HRMs since MSSs are complex systems that fail from time to time, and may become temporarily unavailable. For long lasting jobs accessing many files, which are typical of scientific applications, it is prohibitive to abort and restart a job. Typically, the burden of dealing with an MSS's temporary failure falls on the client. Instead, an HRM can insulate clients from such failures, by monitoring the transfer to the HRM's disk, and if a failure occurs, the HRM can wait for the MSS to recover, and re-stage the file. All that the client perceives is a slower response. Experience with this capability was shown to be quite useful in real situations [2].

A fourth advantage is that SRMs can transparently deal with network failures. SRMs can monitor file transfers, and if failures occur, re-try the request. They can provide clients the information of such failures, so that clients can find other alternatives, such as getting the file from its original archive if a transfer from a replication site failed. Recently, there is an interest of managing the inherent unreliability of the network as part of an extended middleware file transfer service, called "Reliable File Transfer" (RFT). It is intended as a service layer on top of GridFTP that will try to re-transfer files in case of temporary failures of the network, will queue such requests, and will provide status of the requests. When such services are available, SRMs can take advantage of them. Otherwise, as is the case for systems that have no grid middleware software (e.g. only FTP), SRMs need to protect the clients from unreliable network behavior.

A fifth advantage of SRMs is that they can enhance the efficiency of the grid, eliminating unnecessary file transfers by sharing files. As mentioned above, it is typical of scientific investigations that multiple clients at the same site use overlapping sets of files. This presents an opportunity for the SRM at that site to choose to keep the most popular files in its disk cache longer, and providing clients with files that are already in the disk cache first. Managing this behavior is referred to as a "replacement policy", that is deciding dynamically which file to replace when space is needed. This problem is akin to "caching algorithms", which have been studied extensively in computer systems and web caching. However, unlike caching from disk to main memory, the replacement cost in the grid can be quite high, as files have to be replaced from remote locations and/or from tertiary storage. Deploying efficient replacement policies by the SRMs can lead to significant reductions in repeated file transfers over the grid.

Finally, one of the most important advantages of using SRMs is that they can provide a "streaming model" to the client. That is, they provide a stream of files to the client programs, rather than all the files at once. This is especially important for large

computing tasks, such as processing hundreds, or even thousands of files. Typically, the client does not have the space for the hundreds of files to be brought in at once. When making such a request from an SRM, the SRM can provide the client with a few files at a time, streaming of files as they are used and released. This is managed by the SRM enforcing a quota per client, either by the amount of space allocated and/or by the number of files allocated. As soon as files are used by the client and released, the SRM brings in the next files for processing in a streaming fashion. The advantage to this "streaming model" is that clients can set up a long running task, and have the SRM manage the streaming of files, the pre-staging of files, the dynamic allocation of space, and the transferring of files in the most efficient way possible.

6. "Pinning" and "two-phase pinning"

The concept of *pinning* is similar to locking. However, while locking is associated with the *content* of a file to coordinate reading and writing, pinning is associated with the *location* of the file to insure that a file stays in that location. Unlike a lock, which has to be released, a "pin" is temporary, in that it has a time-out period associated with it, and the "pin" is automatically released at the end of that time-out period. The action of "pinning a file" results in a "soft guarantee" that the file will stay in a disk cache for a pre-specified length of time. The length of the "pinning time" is a policy determined by the disk cache manager. Pinning provides a way to share files that are not permanently assigned to a location, such as replicated files. This permits the dynamic management and coordination of shared disk caches on the grid. Since we cannot count on pins to be released, we use the pinning time-out as a way to avoid pinning of files forever.

Two-phase pinning is akin to the well known "two-phase locking" technique used extensively in database systems. While two-phase locking is used very successfully to synchronize writing of files and to avoid deadlocks, two-phase pinning is especially useful to synchronize requests for multiple files *concurrently*; that is, if the client needs several files at the same time, it can first attempt to incrementally pin these files, and only then execute the transfers for all files, then releasing them as soon as each is transferred. We note, that even if file replicas are read-only, a deadlock (or pin-lock) as a result of pinned files can occur if we allow requests for multiple files concurrently. However, if we assume that file requests are asynchronous and that time-outs to release files are enforced, pin-locks are eventually resolved because pinned files are released after they time-out. Nevertheless, two-phase pinning is a useful technique to avoid system thrashing by repeatedly pinning and pre-emptying pins. It requires coordination between the SRMs.

7. The design of "Read" and "Write" functionality of SRMs

When a request to read a file is made to an SRM, the SRM may already have the file in its cache. In this case it pins the file and returns the location of the file in its cache. The client can then read the file directly from the disk cache (if it has access permission), or

can copy or transfer the file into its local disk. In either case, the SRM will be expected to pin the file in cache for the client for a period of time. A well-behaved client will be expected to "release" the file when it is done with it. This case applies to both DRMs and HRMs.

If the file is not in the disk cache, the SRM will be expected to get the file from its source location. For a DRM this means getting the file from some remote location. For an HRM, this means getting the file from the MSS. This capability simplifies the tasks that the client has to perform. Rather than return to the client with "file not found", the SRM provides the service of getting the file from its source location. Since getting a file from a remote location or a tape system may take a relatively long time, it should be possible for the client to make a non-blocking request. To accommodate this possibility the SRMs provide a callback function that notifies the client when a requested file arrives in its disk cache and the location of that file. In case that the client cannot be called back, SRMs also provide a "status" function call that the client can use to find out when the file arrives. The status function can return estimates on the file arrival time if the file has not arrived yet.

HRMs can also maintain a queue for scheduling the file staging from tape to disk by the MSS. This is especially needed if the MSS is temporarily busy. When a request to stage a file is made, the HRM checks its queue. If the HRM's queue is empty, it schedules its staging immediately. The HRM can take advantage of its queue to stage files in an order optimized for the MSS. In particular, it can schedule the order of file staging according to the tape ID to minimize tape mounts and dismounts, as described in [2]. Like a DRM, the HRM needs to notify the client that the file was staged by issuing a callback, or the client can find that out by using "status".

A request to "write" a file requires a different functionality. In the case of a DRM, if the file size is provided, then that space is allocated, and the client can write the file to it. If the file size is not provided, a large default size is assumed, and the available space is adjusted after the file is written. In the case of an HRM, the file is first written to its disk cache in exactly the same way as the DRM description above. The HRM then notifies the client that the file has arrived to its disk using a callback, then it schedules it to be archived to tape by the MSS. After the file is archived by the MSS, the SRM notifies the client again using a callback. Thus, the HRM's disk cache is serving as a temporary buffer for files being written to tape. The advantage of this functionality by HRM is that writing a file to a remote MSS can be performed in two stages: first transferring the file to the HRMs disk cache as fast as the network permits, and then archiving the file to tape as a background task. In this way the HRM can eliminate the burden from the client to deal with a busy MSS as well as dealing with temporary failures of the MSS system.

One of the practical implementation problems that SRMs have to deal with is an incorrect or missing file size. In both cases of getting or putting a file into the SRM space, the SRM needs to allocate space before the transfer of the file into its disk cache. If the file

size provided (or assigned by default) is smaller than the actual file size, then this can cause various failures, such as writing over other files, or overflowing the total space that the SRM manages. There are various methods of dealing with this problem (such as interrupting the transfer or permitting incremental growth of the allocated space), but all require the dynamic monitoring of the file transfers, and the ability to terminate the transfer process if necessary. Since SRMs cannot terminate the transfer process initiated by the client (in the case that it puts a file into the SRM's disk cache), this problem presents a special challenge. The solution to this problem usually requires modifications to the file transfer server program.

SRMs can also be used to coordinate a third party file movement. Essentially, an SRM in site Y can be asked to "pull" a file form site X. This request can be made by a client in a third location. The SRMs in the two sites X and Y then coordinate space allocation, file pinning, and file release. The actual transfer of the file is a regular two-way file transfer from X to Y. The usefulness of this functionality is for clients that produce files, store then temporarily in some location X, and then request their movement to an archive in site Y. The inverse functionality can also be provided, where the SRM at site X is asked to "push" the file to site Y.

8. Conclusion

We discussed in this paper the concept of Storage Resource Managers (SRMs), and argued that they have an important role in streamlining grid functionality and making it possible for storage resources to be managed *dynamically*. While static management of resources is possible, it requires continuous human intervention to determine where and when file replicas should reside. SRMs make it possible to manage the grid storage resources based on the actual access patterns. In addition, SRMs can be used to impose local policies as to who can use the resources and how to allocated the resources to the grid clients. We also introduced the concept of "pinning" as the mechanism of requesting that files stay in the storage resource until a file transfer or a computation takes place. Pinning allows the operation of the coordinated transfer of multiple files to be performed as a "2-phase pinning" process: pin the files, transfer, and release pins. We have developed several versions of prototype SRMs and used them in test cases as part of the Particle Physics Data Grid (PPDG) and Earth Science Data Grid (ESG) projects. A prototype of an HRM was also developed at Fermi National Accelerator Laboratory which interfaces to their Enstore MSS. In addition, efforts are now underway to coordinate the SRM functionality across several projects, including the development of an HRM at Thomas Jefferson National Accelerator Facility to interface to their JASMine MSS, and the European Data Grid to interface to their CASTOR MSS. The emerging concepts and interfaces seem to nicely complement other grid middleware services being developed by various grid projects, such as providing efficient and reliable file transfer, replica catalogs, and allocation of compute resources.

Acknowledgements

We would like to thanks our colleagues John Wu, and Vijaya Natarajan, who provided the bit-map index and the monitoring tool display program for the SC 2001 demo. We also acknowledge the useful interactions with people involved in the PPDG and ESG projects, as well as the European Data Grid project. This work was supported by the Office of Energy Research, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

References

[1] The Grid: Blueprint for a New Computing Infrastructure, Edited by Ian Foster and Carl Kesselman, Morgan Kaufmann Publishers, July 1998.

[2] Access Coordination of Tertiary Storage for High Energy Physics Application, L. M. Bernardo, A. Shoshani, A. Sim, H. Nordberg (MSS 2000).

[3] Particle Physics Data Grid (PPDG), http://www.ppdg.net/

[4] The Grid Physics Network (GriPhyN) http://www.griphyn.org

[5] Earth Science Grid (ESG), http://www.earthsystemgrid.org

[6] Ian Foster, Carl Kesselman, Steven Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organization, *The International Journal of High Performance Computing Applications*, 15(3), (2001) 200-222.

[7] DAGMAN, part of the Condor project, http://www.cs.wisc.edu/condor/manual/v6.2/2_10Inter_job_Dependencies.html

[8] The Globus Project, http://www.globus.org

[9] B. Allcock, A. Chervenak, E. Deelman, R. Drach, I. Foster, C. Kesselman, J. Lee, V. Nefedova, A. Sim, A. Shoshani, D. Williams, High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies, *Proceedings of Supercomputing Conference* (2001).

[10] A. Shoshani, L. M. Bernardo, H. Nordberg, D. Rotem, and A. Sim, Multidimensional Indexing and Query Coordination for Tertiary Storage Management, Statistical and Scientific Database Management Conference (1999) 214-225.

Storage Area Networks and the High Performance Storage System

Harry Hulen and Otis Graf
IBM Global Services
1810 Space Park Drive
Houston TX 77058
Hulen: +1-281-488-2473, hulen@us.ibm.com
Graf: +1-281-335-4061, ofgraf@us.ibm.com

Keith Fitzgerald and Richard W. Watson
Lawrence Livermore National Laboratory
7000 East Ave.
Livermore CA 94550-9234
Fitzgerald: +1-925-422-6616, kfitz@llnl.gov
Watson: +1-925-422-9216, dwatson@llnl.gov

Abstract
The High Performance Storage System (HPSS) is a mature Hierarchical Storage Management (HSM) system that was developed around a network-centered architecture, with client access to storage provided through third-party controls. Because of this design, HPSS is able to leverage today's Storage Area Network (SAN) infrastructures to provide cost effective, large-scale storage systems and high performance global file access for clients. Key attributes of SAN file systems are found in HPSS today, and more complete SAN file system capabilities are being added. This paper traces the HPSS storage network architecture from the original implementation using HIPPI and IPI-3 technology, through today's local area network (LAN) capabilities, and to SAN file system capabilities now in development. At each stage, HPSS capabilities are compared with capabilities generally accepted today as characteristic of storage area networks and SAN file systems.

1. Introduction

Storage Area Network (SAN) technology has a bright future as measured by its growing market acceptance. Web information source allSAN.com [10] reports that:

> Within the mainframe arena, SANs already represent upwards of 25% of data center traffic. Outside of the mainframe area, SANs are expected to account for 25% of external disk storage and approximately 50% of multi-user tape storage by 2003

We believe that SAN technology will only reach its full potential when it can be used to provide secure sharing of data between heterogeneous client systems. To realize this potential requires appropriate storage system software and hardware architectures. One use for such a capability is a SAN-based global file system. A generic host-based file

system provides capabilities such as a naming mechanism, data location management, and access control. A global file system extends this capability to multiple independent operating systems by using specialized protocols, locking mechanisms, security mechanisms, and servers to provide device access. A SAN-based global file system is distinguished from other global file systems by the characteristic that client computers access storage devices directly, without moving data through a storage server.

The High Performance Storage System design and implementation are focused on hierarchical and archival storage services and therefore are not intended for use as a general-purpose file system. HPSS is nevertheless a file system, and specifically, a global file system. While any client applications (such as a physics code) can access HPSS devices with normal Unix-like calls to the HPSS client API library, in normal operation these applications are data transfer applications that transfer data between HPSS files and the local file system. HPSS has a network-centered architecture that separates data movement and control functions and offers a secure, global file space with characteristics normally associated with both LAN-based and SAN-based architectures.

Figure 1 illustrates a typical deployment of HPSS. Note in particular the separation of control and data transfer networks (which may be physical or logical). This inherent separation of control and data helps enable HPSS to present a secure, scalable, global file system image to its users and leads naturally to full global SAN file system capabilities in the near future. The terms "Mover" and "Core Server" in Figure 1 are fairly descriptive of their function, but they are more fully described in Section 5.

This paper tracks the development of concepts and implementation for the separation of control and data functions in storage systems and the importance of these concepts for SAN file systems. These concepts are rooted in work that began over two decades ago [9] and prototyped a decade ago in the National Storage Laboratory (NSL) [3]. Lessons learned at the NSL led to the architecture of the High Performance Storage System (HPSS), which today supports a variety of high-speed data networks [4, 5]. HPSS is a collaborative development whose primary partners are IBM and the U.S. Department of Energy. This collaboration has been in existence for a decade, and HPSS development is ongoing. We discuss simple extensions to HPSS to exploit today's SAN technology within large-scale HSM storage systems. We conclude with a section on lessons learned.

2. SAN Terminology

Several definitions of a Storage Area Network exist as related to common, shared repositories of data. The Storage Networking Industry Association (SNIA) online dictionary offers the following definition of Storage Area Network [1]:

1. A network whose primary purpose is the transfer of data between computer systems and storage elements and among storage elements. Abbreviated SAN. SAN consists of a communication infrastructure, which provides physical connections, and a management layer, which organizes the connections, storage elements, and computer systems so that data transfer is secure and robust. The

term SAN is usually (but not necessarily) identified with block I/O services rather than file access services.

2. A storage system consisting of storage elements, storage devices, computer systems, and/or appliances, plus all control software, communicating over a network.

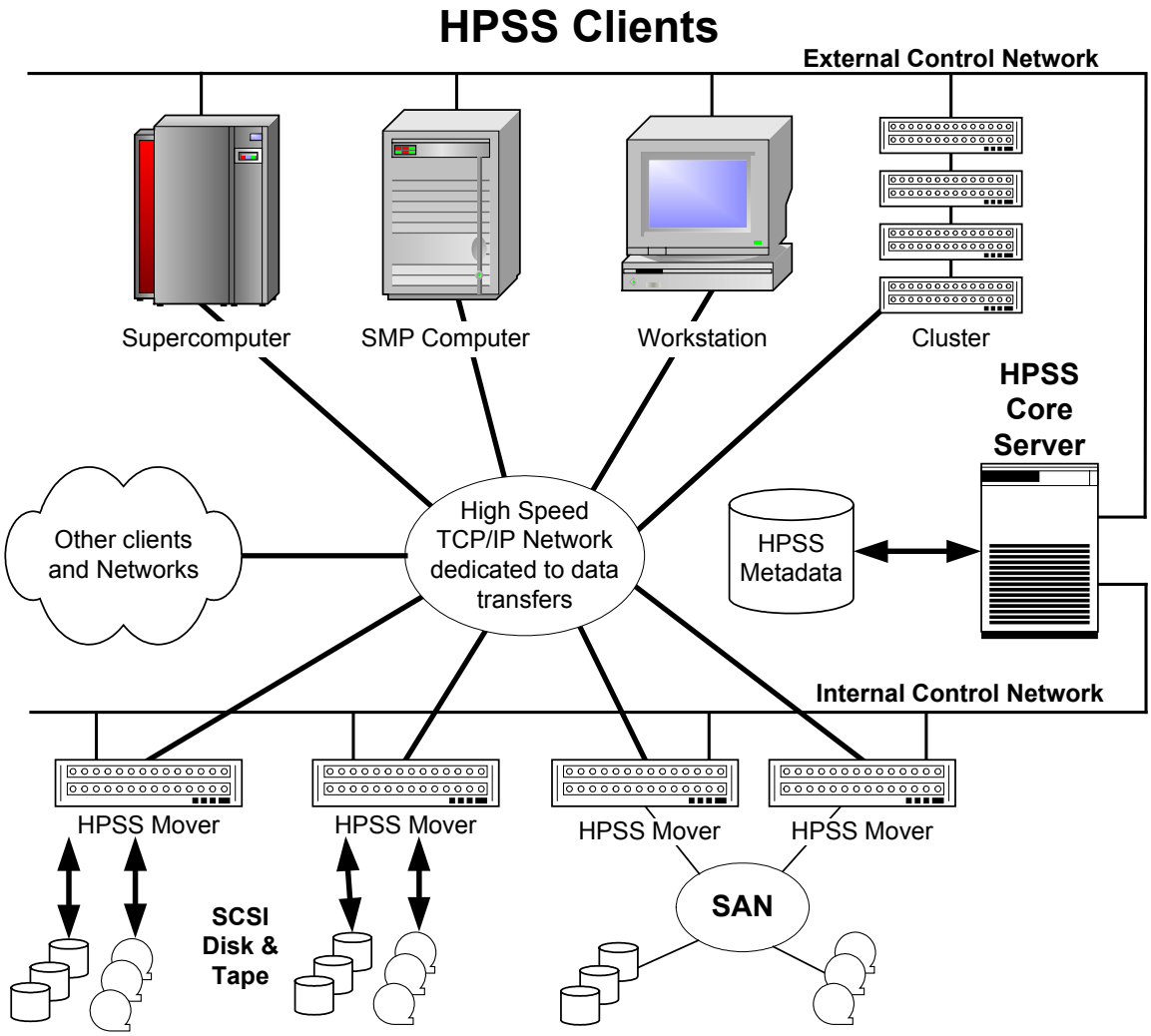


Figure 1: HPSS storage systems support a network centered architecture

Our interest is in large, high performance storage systems where 100s – 1000s of terabytes of data can be shared among client computers. The focus of SANs in our paper is from Bancroft et al [2]:

> The implementation [of a SAN] permits true data and/or file sharing among heterogeneous client computers. This differentiates [SAN file systems] from SAN systems that permit merely physical device sharing with data partitioned (zoned) into separate file systems. … The software orchestrating the architecture is what unites the components and determines exactly how these elements behave as a system.

The same paper defines the notion of a SAN file system. Figure 2 illustrates the control and data flow of a such a generic SAN file system.

> The optimum vision is a single file system managing and granting access to data in the shared storage with high bandwidth Fibre Channel links [today there are other network technologies] facilitating transfers to and from storage. … The objective … *is to eliminate file servers* between clients and storage with minimum or no impact to the controlling applications. *Control information is typically separated from data traffic and in some architectures the two are isolated on completely separate networks.*

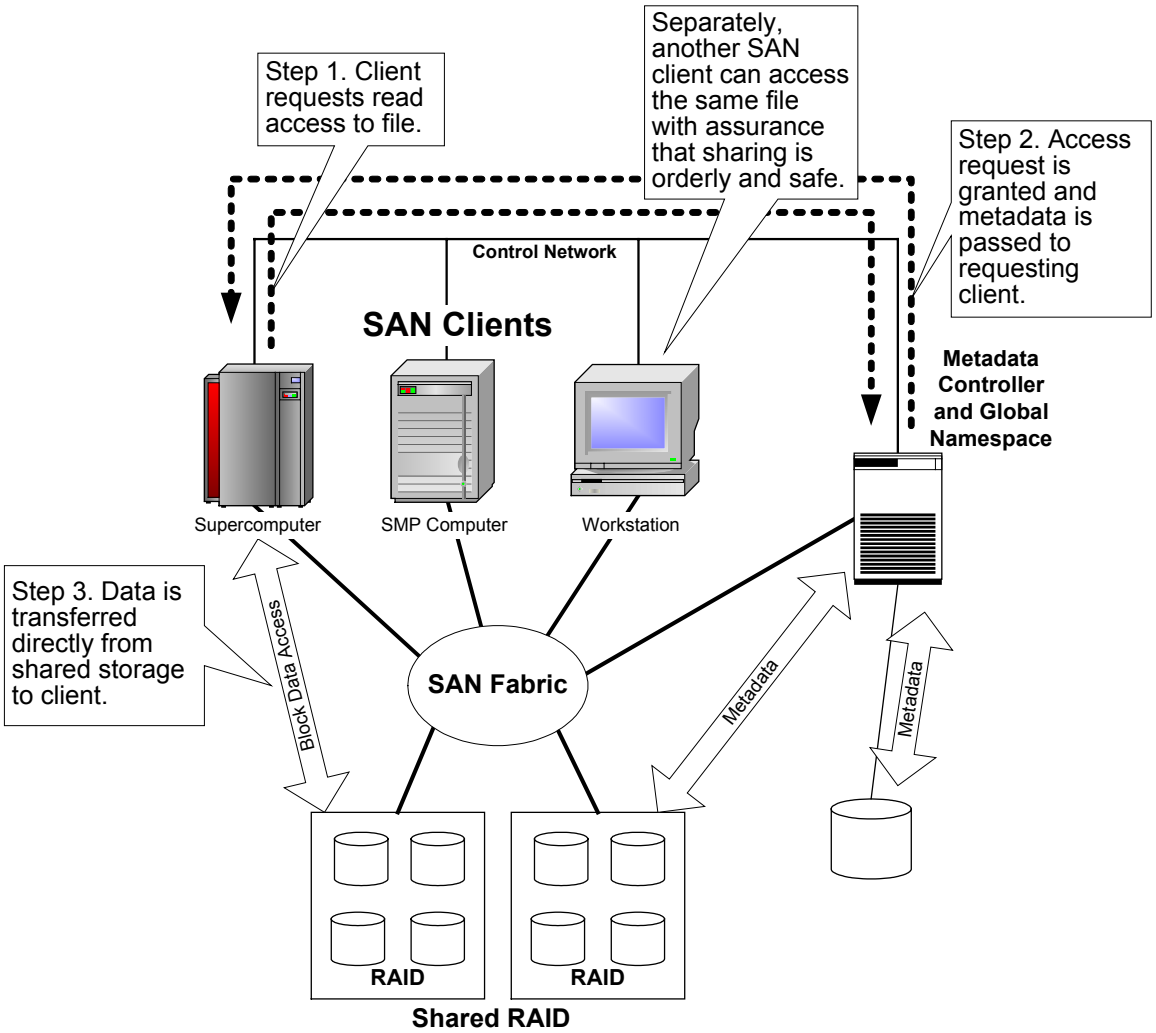


Figure 2: A file read operation illustrates the separation of data and control in a typical SAN file system.

It will be shown in the following sections that HPSS current implementation incorporates significant components of the SAN file system functionality described in the above definition, and how additional SAN file system functionality will be added to HPSS.

3. SAN Precursors

Although the term "SAN" is relatively new, the basic ideas of shared file systems have been around since the early days of computing. Papers by Thornton [8] and Watson [9] trace shared file concepts to the Octopus network at Lawrence Livermore National Laboratory in the 1960s, the Network Systems Corporation Hyperchannel, and the IEEE Mass Storage Reference Model in the late 1970s and early 1980s.

The foundation for HPSS can be traced to 1992 and the National Storage Laboratory (NSL). The NSL was a joint government/industry collaboration investigating high performance storage system architectures and concepts [3]. Work at the National Storage Lab led to NSL-Unitree, a prototype hierarchical storage system incorporating a distributed storage architecture that leveraged third-party data transfers almost a decade in advance of today's SAN deployments. A third-party data transfer is a data transfer controlled by an agent. The agent controls the data transfer by communicating with both the data source and the data sink in setting up the transfer. The agent does not participate in the actual movement of the data.

MAXSTRAT Corporation, a partner in the National Storage Lab, built high-end HIPPI-based RAID devices known as Gen4 and Gen5 arrays. These disk arrays were among the highest performing RAID disk devices of their day. Using the IPI-3 protocol, NSL-Unitree was able to achieve data rates of about 60 MB/s between a Cray C90 and MAXSTRAT disks over a HIPPI network.

IPI-3 was the third release of the Intelligent Peripheral Interface, a standards-based I/O interface that at the time was considered to be a high-end alternative to SCSI. Like SCSI, IPI-3 could exist as a native physical level protocol, or it could be encapsulated and sent over another general-purpose protocol such as HIPPI framing protocol. Disks were available equipped with a native IPI interface. Both IPI and TCP/IP could coexist on a HIPPI network through the use of HIPPI framing protocol.

The MAXSTRAT disk array was connected to a high performance computer via parallel or serial HIPPI, which has a nominal data rate of 100 megabytes per second. Originally designed as a point-to-point parallel interface, HIPPI evolved to be a switchable serial interface using a fibre transmission medium. Through the use of HIPPI switches, the Gen5 could be connected to multiple computers. By using encapsulated IPI, each computer could communicate with any Gen5 disk array as though it were a local IPI-3 device. Today this would be analogous to sharing a Fibre Channel disk array using SCSI over Fibre Channel, or more recently Gigabit Ethernet with SCSI over IP.

Significantly, the Gen4 and Gen5 implemented the third-party capabilities of the IPI-3 standard. With this capability, IPI-3 commands could be sent to a central server that mediated the requests and redirected them to source and sink for third-party transfer to bring order and preserve data integrity. The following description of the third-party architecture from Chris Wood [6]:

Third-party transfer architectures address the data "ownership" and access control issues by consolidating all data ownership and file system knowledge in a centralized server. Unlike NFS-style architectures, third-party transfer allows for direct disk I/O access to the central data store by clients. This architecture eliminates the burden of heavy inter-host lock manager and semaphore traffic and presents a well understood, NFS-like application interface. User data flows at local disk speeds (vs. network speeds) over dedicated high-speed disk channels while control traffic flows over a separate control network. The goal is to deliver data at optimal speeds with no interruptions for read/write commands and flow-control handshaking.

Essentially, The NSL proved the basic concepts of what we would now call a SAN file system. Figure 3 illustrates a file read operation in the NSL prototype. Note that Figure 3 is almost identical with Figure 2. Details of the protocol operation are given in [3].

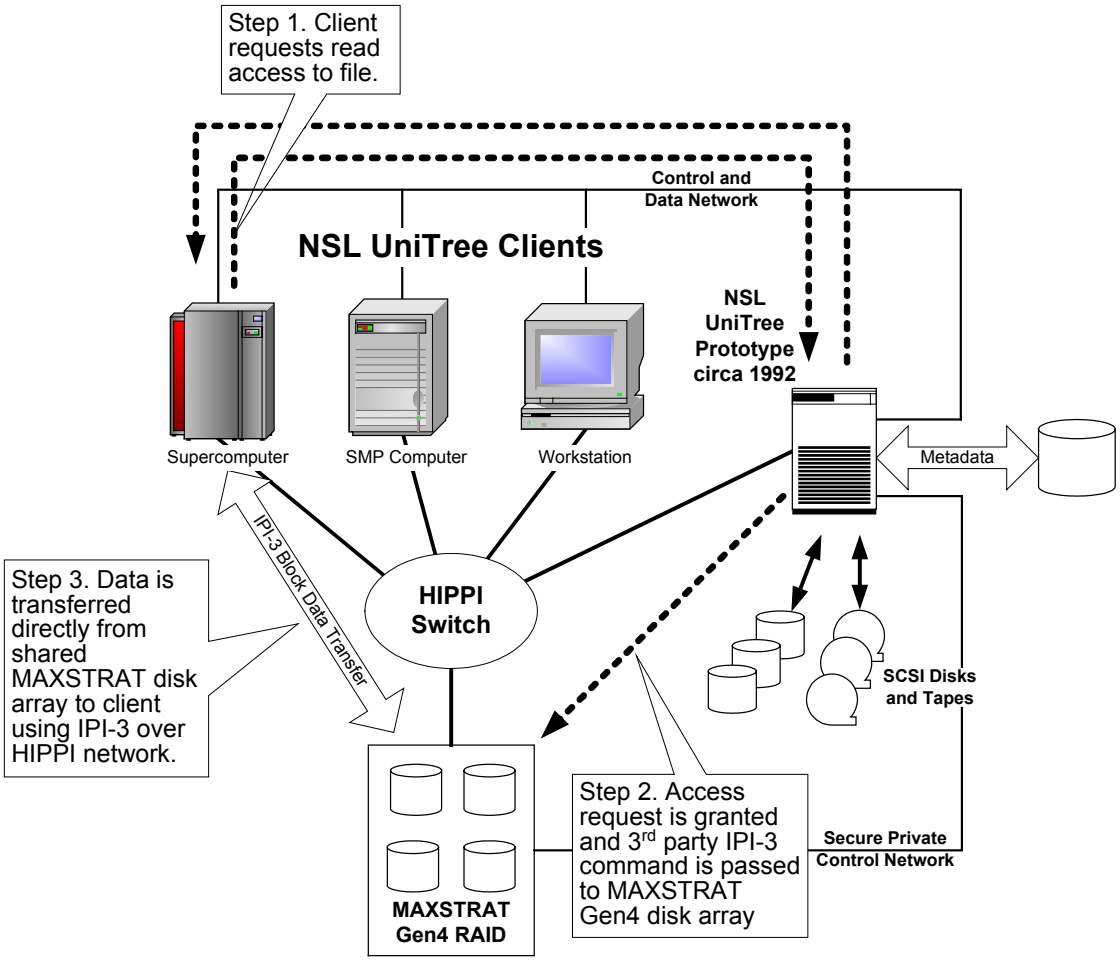


Figure 3: The NSL Prototype provided 3rd party "LAN-less" data transfers.

The NSL prototype proved several points to the NSL collaboration:

1. It established that data transfers between a client and network attached disks could give as good or better performance as native client disk.

2. Third-party data transfer allowed the transformation of the NSL server to function as a metadata engine that could effectively control data sharing among clients while maintaining high data rates.
3. Security is aided by separating control and data flow to separate networks.
4. Hierarchical storage, with movement of data between disk and tape, could be implemented in the shared disk environment.

4. Security Implications for SAN File Systems

Whenever data is shared among multiple clients, effective security mechanisms must be provided. In the case of robust global storage systems, security has historically been enforced by the file or storage server that effectively isolates clients from storage devices. NFS v4, AFS, DFS, and HPSS are examples of global storage systems that offer authenticated and authorized transfers between the client and storage servers. However when you make storage devices directly accessible to client systems, as in today's SANs, you have in effect opened a "Pandora's box" of security problems.

In today's SAN environments, shared storage appears as directly accessible devices on every client requiring access to the shared data. The level of protection for a shared SAN device is therefore no stronger than it would be for a local device attached to the client. This means that if any SAN client machine is compromised at the operating system root level, all shared data has been compromised. In effect, all shared-storage clients need to trust each other. SAN zoning limits visibility of devices to specified hosts and can be used to protect data by limiting access. But in cases where the goal is to make data globally accessible to many clients, security risks are incurred if any but the most trusted clients are added.

The NSL developers recognized this issue and provided a reasonable level of security by using a secure private control network connection between the storage servers and the network attached storage devices (See Figure 3). The storage system controlled access to all shared data. Clients did not have direct access to the storage devices because of the nature of the IPI-3 third-party protocol. Access to a network connection was granted to processes running on the storage clients on a per-transfer basis. The storage system used the secure private network to communicate with the MAXSTRAT disks, acting as the third-party agent facilitating all transfers between the storage clients and the network attached peripherals. It would have been very difficult for a rogue client to compromise the security of the NSL storage environment with this mechanism.

A similar level of security must be developed for use in a current SAN environment before the true power of SAN file systems can be safely realized. Object based "Network-Attached Secure Disks" [7] could solve this problem if they are accepted within the storage marketplace.

5. The Development of HPSS

The HPSS collaboration [4, 5] took up the work of the National Storage Laboratory collaboration in 1992 under a Cooperative Research and Development Agreement (CRADA) between IBM and several U.S. Department of Energy Laboratories (Lawrence

Livermore, Los Alamos, Oak Ridge, and Sandia). After reviewing the projected requirements of next generation high performance HSM systems and all available hierarchical storage systems then in existence, the collaboration concluded that it was necessary to develop new software that would provide a highly scalable storage system, anticipating the growth in data-intensive computing (100s – 1000s of terabytes and Gigabyte/sec data transfer rate ranges) while also providing robust security for global file access. As this was to be a collaborative development, there was need for open access to source code among all collaboration members. The first production release of HPSS was in 1995, with major releases since then at approximately one-year intervals. Development is ongoing, with about 28 full time equivalent developers, including about 16 in the Department of Energy labs. Ongoing development is discussed in later sections. There are currently over 40 production HPSS sites worldwide in government, research, and education.

The scalability requirement led to a network-centered architecture that allowed more storage capacity and increased data rates by adding management and storage elements to a scalable network. Like the earlier NSL prototype, HPSS was designed to accommodate intelligent third-party devices based on the model of the MAXSTRAT Gen4 and Gen5 disk arrays [4]. It was assumed that more intelligent third-party devices would follow; however, it was recognized that most of the storage devices that would be attached to HPSS would be conventional disks, disk arrays, and tape libraries. To accommodate conventional devices, the HPSS collaboration introduced the idea of a "Mover". The notion was to attach SCSI disks and tape drives to low-cost computers running a lightweight HPSS Mover protocol. A data Mover and the disks and tapes attached to it formed the equivalent of an intelligent third-party device. Thus the HPSS architecture enabled both ordinary and intelligent devices and reasonably priced computers to work together while preserving security and a global name space.

Figure 4 illustrates the network-centered data flow of HPSS for a file read operation. Comparing this figure with the previous NSL illustration (Figure 3), one can see that the Mover and the disks and tape drives attached to it take on the attributes of an intelligent third-party device.

The HPSS Core Server presents the image of a file system to the user. Its main function is to manage the client interface and the system's metadata (e.g. data location and security data). At the lower level involved with data transfer, the lightweight HPSS Mover code works only with block I/O. Unlike conventional network-attached storage (NAS), HPSS Movers transfer data over the network at a block level, not a file level, simulating the low-level I/O of early intelligent third-party devices and today's SAN-attached devices. The Mover is strictly an intermediary to transfer logical blocks of data under control of the HPSS Core Server. See references [3, 4, 5] for details.

Use of multiple Movers allow many concurrent data transfers to provide very high aggregate data transfer rates. HPSS also supports data striping (parallel data transfers), thereby providing very fast single file transfer rates [4].

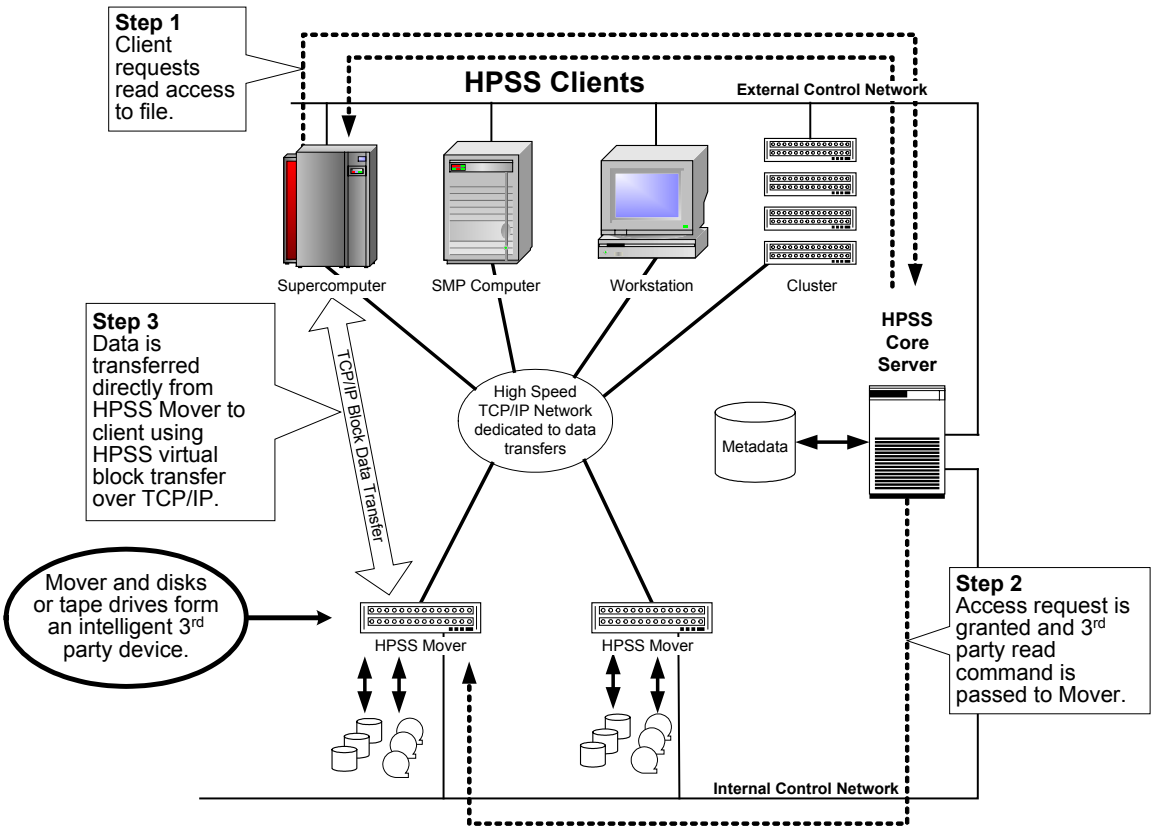


Figure 4: HPSS Movers create third-party capability using conventional devices.

HPSS, with its network-centered, third-party architecture is well suited to leverage SAN technology. The next section explains how SAN technology is used with HPSS today, and the sections that follow show enhancements will further exploit SAN technology.

6. Today's SANs and HPSS

Today's SAN technology promises better management and sharing of storage devices across HPSS Movers. SAN technology can simplify administration of large amounts of storage and can lead to better system reliability.

HPSS LAN-based configurations (refer back to Figure 1) are capable of providing very high bandwidths, both for individual data transfers and in the aggregate across concurrent file transfers and can furthermore support parallel, striped data transfers across multiple disks or tape drives. The current HPSS Mover architecture allows devices to be run at data transfer rates equal to 85% to 95% of the best possible device data transfer rates achievable at the block I/O level. Inexpensive network technologies such as Gigabit Ethernet, together with more efficient TCP/IP protocol implementations assure that LAN-centered technology is neither a performance bottleneck nor a cost issue for today's HPSS sites. Moore's Law has made Mover hardware inexpensive for lower I/O rate devices such as tapes but for high throughput disk environments (100s MB/s per Mover) Movers are still relatively expensive. Thus, neither initial cost nor performance are sole motivators for introducing SAN technology into HPSS in some environments. For those

requiring Movers capable of highest I/O rates, cost may be a motivator. SAN capabilities are important to the HPSS community because they will allow users of HPSS much more flexibility to reconfigure disks and tape drives when needs change.

The ability to reconfigure is especially important in case of component failures, including network, Mover, and device components. With SAN technology, disks and tape drives can be quickly reallocated among Movers, allowing quick restoration of service. Going one step further, SAN technology enables disks and tape drives to be connected to pairs of HPSS Movers, allowing the use of fault-tolerant software such as IBM's High Availability Cluster Multi-Processing (HACMP). All of these capabilities are available with today's HPSS just as they are available with other storage software, because SAN technology presents computers with the image of local disks or tape drives. Our goal is to exploit SAN technology as the high performance network connecting both clients and devices. This allows clients direct access to SAN devices, saving network store and forwards and data copies. Above the SAN level of device sharing and reconfiguration, HPSS adds the capabilities of a hierarchical, shared file system.

Having looked at how HPSS sites use SAN technology today to aid system administration and recovery from component failures, we now show how SAN capability will be exploited in future releases of HPSS.

7. SAN-enabled Movers and Clients

We have set a course to enable client applications to read and write data directly over a SAN, bypassing the existing store and forward character of TCP/IP networks when used with SCSI devices. In doing so, we will also enable HPSS to read and write data directly over a SAN for internal purposes such as migration and staging. The changes create "SAN-enabled Movers" and "SAN-enabled Clients."

We are currently evaluating a prototype that is an extension of the IPI3 I/O redirection mechanism for disk access described earlier in the paper. Devices are assigned to a single Mover as is currently done in HPSS. In the case of I/O between a SAN-attached disk device and a SAN-attached client, the SAN-enabled disk Mover redirects its I/O descriptor (an internal HPSS data structure) to the client, which in turn can perform the I/O operation directly with the SAN disk. The "client" in this case could be either a true HPSS Client (i.e. a user) or another Mover such as a tape Mover. No data passes through the disk Mover, as it is only used for the redirection control. Only a single disk Mover or a small number of disk Movers would be required, reducing cost. This design is called "I/O Redirect Movers."

We are also studying a design that allows HPSS to dynamically map a device to the a Mover for a data transfer. This design is called "Multiple Dynamic Movers." Currently devices are administratively assigned to specific Movers. With Multiple Dynamic Mover capability, it will be possible to configure SAN-enabled Movers and Clients that are equivalent to the I/O Redirect Mover capability in data transfer functionality and offer dynamic device to Mover mapping, which may be useful for dynamic failure recovery

and load balancing. In the case of Clients, this would be accomplished by combining a SAN-Enabled Mover with a conventional Client API library.

We will have a prototype of SAN-enabled Movers and Clients running in an HPSS testbed in the spring of 2002. Experience with that prototype and the other design and requirements studies under way will lead to our final implementation choices. The selection of the "I/O Redirect Mover" or the "Multiple Dynamic Mover" will be made by mid year 2002 so as to deliver a SAN-enabled product in 2003. The discussion that follows applies to either approach.

For most systems configured for SAN enablement, fewer Movers will be required. Data transfer across a LAN is avoided. However, SAN enablement of Movers and Clients will be optional, and existing LAN-based capabilities will be fully supported. Sites that elect to use SAN-enabled Movers and Clients will benefit from fewer "hops" between HPSS-managed disk and the user and between disk and tape. On the other hand, the stronger inherent security for shared storage that is afforded by the current HPSS Mover and LAN approaches will in general (independent of HPSS) motivate some sites to use SAN enablement only for HPSS internal functions of migration and staging, while retaining LAN-based client functions. This will be discussed in more detail in Section 10.

Now we look at the ways SAN-enabled Movers and Clients can be exploited. These ways are (1) LAN-less and Server-less data movement for HSM stage and migrate and (2) LAN-less data movement between clients and storage devices directly over the SAN.

8. LAN-less and Server-less Data Movement for HSM Stage and Migrate

The HSM stage/migrate function moves data between levels in the storage hierarchy, usually consisting of disk and tape. In the current HPSS architecture, each storage device is assigned to a single data Mover. Data that is being staged to disk or migrated to tape is transferred between the respective Mover machines over a high-speed TCP/IP network.

SAN architecture is capable of making storage devices directly accessible to all Mover platforms connected to the SAN. With SAN-enabled Mover approaches outlined above, one Mover computer (which may run multiple Mover processes) will have the I/O descriptors for both source and sink ends of the transfer. Thus it will have the capability to migrate data from disk to tape or to stage data from tape to disk without moving data across a LAN. Eliminating a LAN transfer should allow fewer Mover computers and fewer LAN data connections. This is shown in Figure 5.

Going one step further, when devices and clients are directly attached to a SAN, the potential exists for the actual data movement to take place without going through a Mover by using the SCSI third-party copy command from a third-party agent. This capability is used in some tape backup systems today, and the same capabilities can be applied to hierarchical storage. Since the HPSS Mover software in Figure 5 has the addresses of both the disk and tape drive (source and sink), it can be extended to provide this third-party SCSI copy service or use another SAN agent specializing in this service.

We expect to consider this Server-less data transfer capability in the near future and see it as a logical extension to the LAN-less SAN enablement described above.

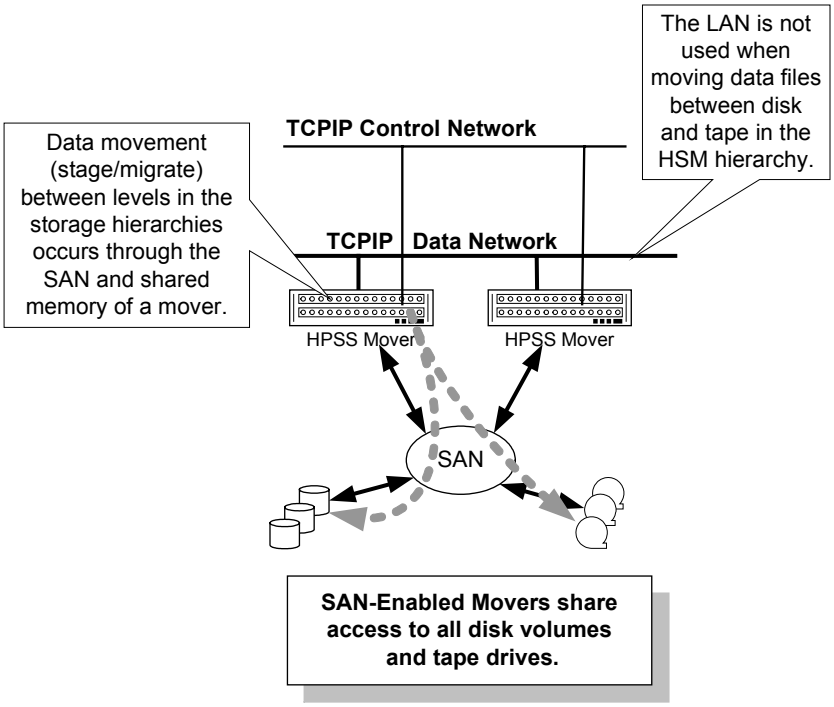


Figure 5: LAN-less Stage/migration between disk and tape using SAN-enabled Movers

9. LAN-less Data Movement between Clients and HPSS Storage Devices

The high performance user interfaces of HPSS are the Client API library, which is a superset of the Unix standard I/O read and write services augmented for parallel I/O, and Parallel FTP (PFTP), which is similarly a superset of Unix ftp. The Client API library, has code to support the Mover protocol and communicates with HPSS Movers using TCP/IP if the client and Mover are on different machines, or by an internal transfer mechanism if they are in the same computer.

SAN-enabled HPSS Clients will be able to access SAN-attached HPSS disks directly, and potentially also SAN-attached tapes. This can be done because the Client will be passed an I/O descriptor that describes the I/O operation to be performed. This is shown in Figure 6. The benefit of a SAN-enabled Client API library on a client machine must be weighed against the security exposure. This is discussed in the next section.

10. Security Considerations for Access to Storage: SAN versus LAN

We will now revisit security issues. Our assumption is that with today's generally available Unix-based technologies, a person who acquires root access, whether with authorization or not, can read and write any disk or tape that is configured as a local device. This includes SAN-attached devices. This is a well-known vulnerability of SANs, and it is the basic reason for zoning. The problem is that zoning and sharing data are inherently at odds with each other. In an environment where access to a computer cannot

be limited by physical means, then the information on shared devices is vulnerable to a rogue user with root access on any SAN-attached machine zoned for access to the shared data. (Zoning is a SAN capability that allows users to create multiple logical subsets of devices within a physical SAN as mentioned earlier. Access to devices within that zone is restricted to the members of the zone.) For this reason and until improved technology such as secure object-based devices [7] are available, server-facilitated access is currently the safest course for a file or storage system shared across computers.

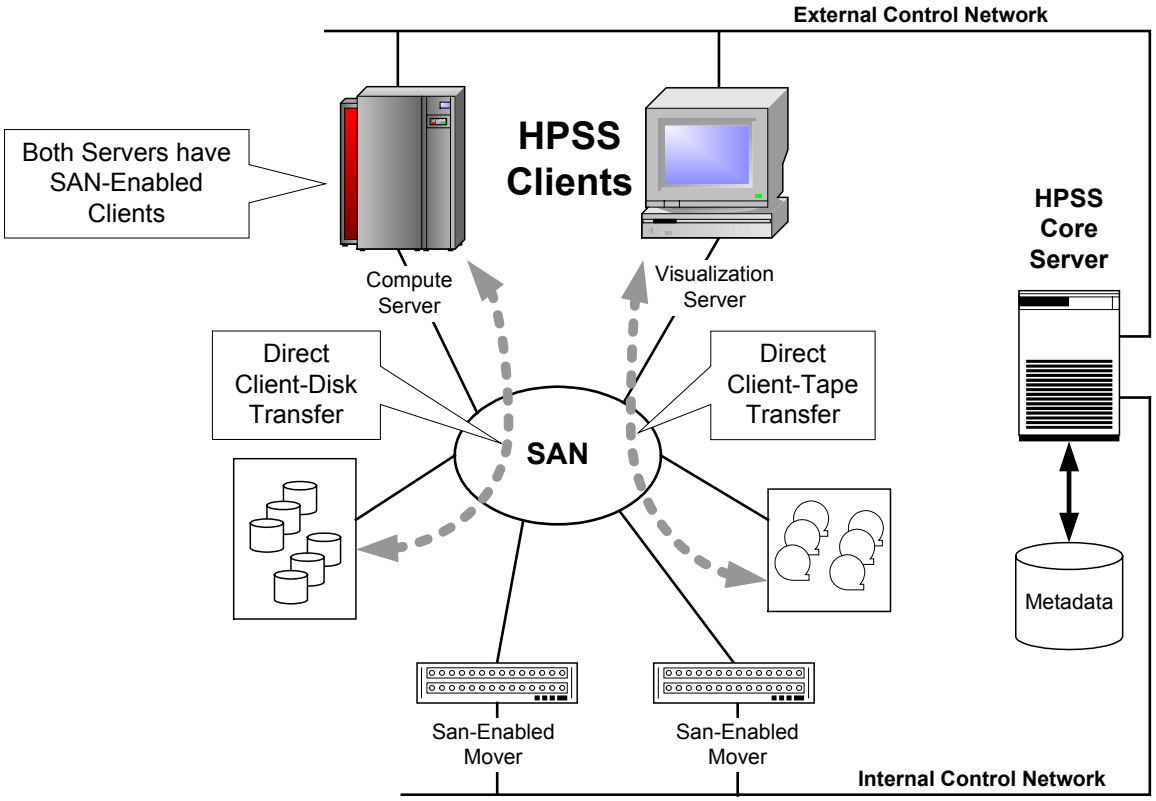


Figure 6: With SAN-enabled Movers and Clients, HPSS has LAN-less access to disk and/or tape storage.

Most large computer centers contain computer systems that are not likely to be compromised, usually because access is limited. For systems where access can be limited and trust exists, then sharing files across computers using SAN devices may present an acceptable level of risk.

Figure 7 shows appropriate use of current SAN and LAN capabilities for an example limited-access computer system and for an example open-access computer system. The configurations shown are typical of large IBM SP computers, large Linux clusters, and similar large-scale distributed architectures. By “limited access” we mean a computer system where access is physically controlled such that rogue users are very unlikely to gain access to the I/O client nodes, while an “open” system would be less secure and the I/O client nodes would be more vulnerable. For simplicity only the data paths are shown in Figure 7. Control would typically be over a fast Ethernet.

Each computer system in the example of Figure 7 has a local file system such as the IBM General Parallel File System (GPFS). GPFS is the principal file system for the IBM SP and is also used with Linux clusters. GPFS as configured here would provide access to files across nodes within each computer system but not across computer systems. Therefore GPFS data accessible to one system would be on disk zones not visible to the other computer system. This is the classic use of SAN zoning to protect each computer system's local file system. Use of SAN zoning to allocate storage to HPSS and local file systems is the heart of the administrative benefit of SANs.

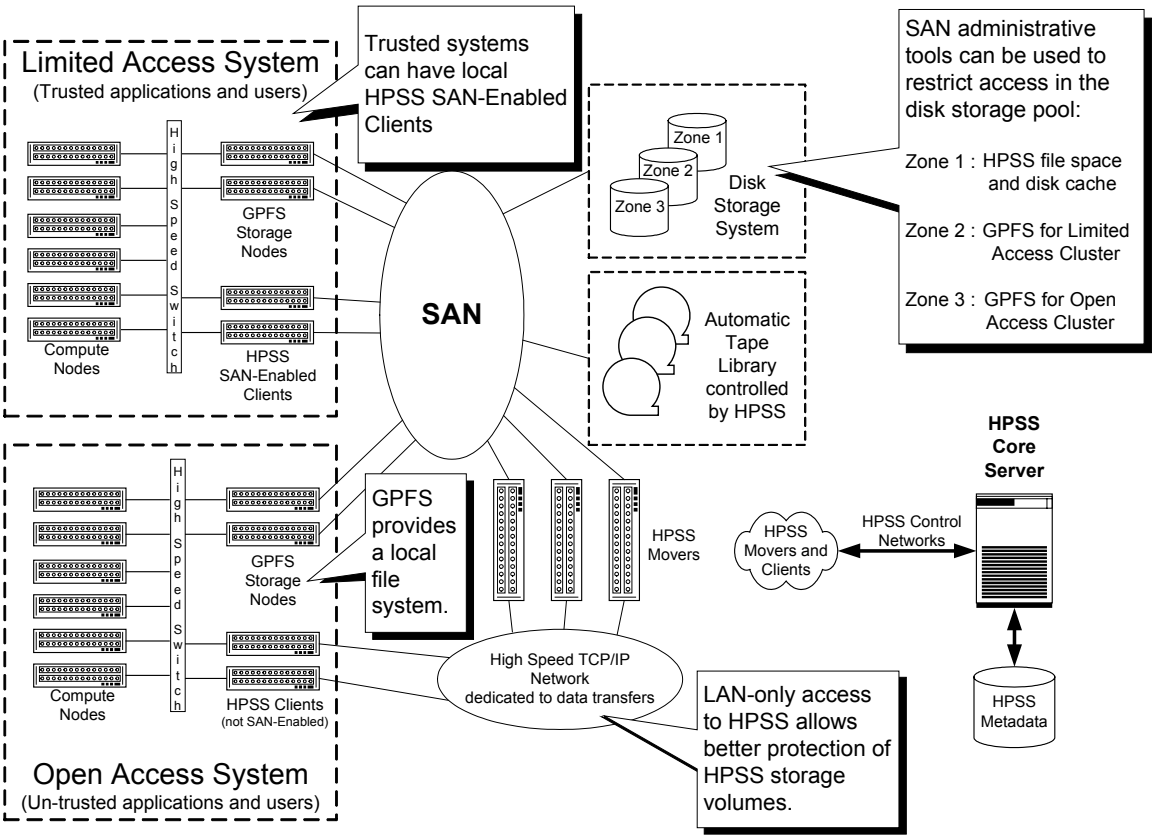


Figure 7: Example of where HPSS provides a global file system to both trusted and un-trusted clients.

HPSS, on the other hand, is typically configured such that files are globally visible across all HPSS client computers (although HPSS clients can be configured with limited access to particular classes of HPSS files). HPSS files in our example are in zones that are visible to all HPSS Client nodes, both in the Limited Access System cluster and in the Open cluster. As a result, data transfers from HPSS to nodes in the Limited Access System cluster will occur over the SAN and no external LAN is required for data transfer. SAN terminology would be "LAN-less" or "LAN-free" transfer.

For a system with a reasonably small number of compute nodes in the cluster it would be possible to put a SAN-enabled Client on each compute node, thereby eliminating the need to transfer data across the backbone network of the cluster. However for a large cluster or SP, this would require an equally large SAN switch. It would also open the

HPSS data zones to the previously described vulnerability of SANs to rogue users with root access to the compute nodes. This vulnerability is not a limitation of HPSS but is due to the lack of security mechanisms to protect shared data in today's SANs. It would therefore be recommended that in most situations, dedicated nodes be used for the HPSS Clients. At LLNL, for example, the normal practice is to use agents to transfer data between HPSS and GPFS, which serves as the local file system. Only the agents are enabled to use the HPSS Client API and PFTP. Residing on SP nodes dedicated to I/O, these agents and the client API are protected from unauthorized access and hence the associated SAN zones are protected from unauthorized use.

The Open Access System, which is the less trusted of the two systems, is configured to access HPSS files only through the LAN, using conventional capabilities of the HPSS Client without SAN enablement. This provides the maximum protection for HPSS data.

11. Lessons Learned

The HPSS collaboration and the earlier NSL collaboration have dealt with the problems of scalable, network-centered storage for over a decade. Our charter is to provide storage software for large, demanding applications such as those of the Department of Energy labs that sponsor HPSS. Other large applications where HPSS has been deployed include supercomputer centers, weather, high-energy physics, and defense. Our "lessons learned" apply both to this high end of hierarchical storage and archiving and we believe to SAN file systems generally. Our experience has led us to a blend of LAN-based and SAN-based technologies with the overarching requirements of scalability, high data rates, shared access to files, security, high availability, and manageability.

Based on our experience with HPSS and our forty plus installations we have found that:

- High data rates and scalability are supported by a network-centered architecture, but not tied to either LAN or SAN.
- The lightweight HPSS Mover, which is based on a concept from the IEEE Mass Storage Reference Model Version 5, is a useful tool for scalability and facilitates simple evolution toward full support for SAN file system concepts.
- LAN-based and SAN-based technologies are complementary and can be mixed.
- Data rates are limited by the hardware configuration (including the network and the choice and number of devices) and not by HPSS software.
- Due to the lack of an adequate SAN security mechanism, shared access to data is best managed in a server-based environment for situations requiring protection from a rogue users who might obtain root access.
- Manageability and high availability are enhanced by SAN capabilities.
- Separation of data network paths from control network paths enhances security.

We find that the blending of LAN and SAN capabilities of current and future releases of HPSS effectively addresses scalability, high data rates, shared access to files, security, availability, and manageability ways that are useful to high-performance data-intensive computing. We believe that the lessons of NSL and HPSS have applicability to others in our industry exploring or developing SAN based file and storage systems, as the current explosion of electronic data goes on around us.

12. Acknowledgements
We wish to thank the early participants in the National Storage Laboratory for their support of early network centered storage architectures and the many developers within the HPSS Collaboration who have created HPSS. This work was, in part, performed by the Lawrence Livermore National Laboratory, Los Alamos National Laboratory, Oak Ridge National Laboratory, National Energy Research Supercomputer Center and Sandia National Laboratories under auspices of the U.S. Department of Energy, and by IBM Global Services - Federal.

References

[1] The Storage Network Industry Association (SNIA) has an excellent dictionary on their web site, www.snia.com. The dictionary is currently located in the "Resource Center" area of the web site. The definitions in this paper differ somewhat from SNIA definitions, but the authors acknowledge the authority of the SNIA dictionary.

[2] M. Bancroft, N. Bear, J. Finlayson, R. Hill, R. Isicoff, and H. Thompson, "Functionality and Performance Evaluation of File Systems for Storage Area Networks (SAN)," *Proceedings Eighth Goddard Conference on Mass Storage Systems*, College Park, MD (Mar 2000). This paper has an excellent overview of SAN file systems.

[3] R. Hyer, R. Ruef, and R. W. Watson, "High Performance Direct Network Data Transfers at the National Storage Laboratory," *Proceedings Twelfth IEEE Symposium on Mass Storage*, Monterey, CA (Apr. 1993). This paper documents the history of NSL-Unitree and 3rd party IPI-3.

[4] R. A. Coyne and R. W. Watson, "The Parallel I/O Architecture of the High Performance Storage System (HPSS)," *Proceedings Fourteenth IEEE Symposium on Mass Storage, Monterey*, CA (Sept. 1995)

[5] D. Teaff, R. W. Watson, and R. A. Coyne, "The Architecture of the High Performance Storage System (HPSS)," *Proceedings Goddard Conference on Mass Storage and Technologies*, College Park, MD (Mar. 1995). For more recent HPSS architectural information, refer to the HPSS web site www.clearlake.ibm.com/hpss.

[6] C. Wood, "It's Time for a SAN Reality Check," available at http://www.maxstrat.com/san_wht.html. This paper includes a discussion of third-party data transfer as implemented in the MAXSTRAT Gen4 and Gen5 disk arrays.

[7] Garth A. Gibson, David F. Nagle, Khalil Amiri, Fay W. Chang, Howard Gobioff, Erik Riedel, David Rochberg, and Jim Zelenka. "Filesystems for Network-Attached Secure Disks" CMU-CS-97-118 July 1997

[8] Thornton, James E., "Back-end Network Approaches", IEEE Computer, Vol. 13, No. 2, Feb. 1980, pp 10 -17. This paper reviews the history of storage network approaches and outlines the directly attached storage features of Hyperchannel.

[9] Watson, Richard W., "Network Architecture Design for Back-End Storage Networks", IEEE Computer, Vol. 13, No. 2, Feb. 1980, pp 32-49. This paper reviews why a shared file system approach is critical to success of storage networks and outlines the architecture that became the IEEE Reference Model, UniTree and HPSS, including third-party transfers, Movers, and direct device to device transfers.

[10] allSAN Research Services, http://www.allsan.com/marketresearch.php3

Introducing A Flexible Data Transport Protocol for Network Storage Applications

Patrick Beng T. Khoo and Wilson Yong H. Wang
MCSA Group, NST Division – Data Storage Institute
Affiliated to the National University of Singapore
Funded by the Agency for Science, Technology And Research Singapore
DSI Building, 5 Engineering Drive 1, 117608 Singapore
Tel: +65 8748413 Fax: +65 7772053
Email: patrick@dsi.nus.edu.sg
Web: http://nst.dsi.nus.edu.sg/mcsa/

Abstract

The purpose of this paper is to demonstrate that alternative solutions to current methods exist for network storage. We would like to introduce one such alternative, a new protocol that we call HyperSCSI. This protocol is used for the transmission of Small Computer Systems Interface (SCSI) family of protocols across a network and multi-technology device support. In this paper, we will outline some of the key features and basic technical details of HyperSCSI. We have also developed several fully functioning disk array prototypes using a variety of hardware and storage devices as well as conducted benchmarks and performance tests on this. A performance comparison between this new protocol and iSCSI and NFS is also included here.

1. The Problem

Research has been ongoing for ways to transport data over networks for storage applications for quite some years. While we pursued efforts in developing network storage technologies, we came across the following issues and concerns.

- High cost of Fibre Channel SANs – Implementing and managing FC-based SANs is quite expensive. Even if hardware costs were to come down (and we expect them to do so), ultimately the "hidden" costs of systems, infrastructure, manpower and software implementation and maintenance is still very high.
- TCP/IP SAN performance is still not good enough without hardware acceleration – TCP/IP is inherently slow compared to FC-based storage technologies. Special hardware for TCP/IP represents higher costs and a more difficult upgrade path for users.
- FC-based SANs cannot do Storage Wide-Area Networks – FC is an inherently local communications technology, and if one needs to go wide-area, the best method is to use the ever present IP due to its wide-spread availability. Ongoing efforts such as FCIP, iFCP and iSCSI are in line with this idea.
- Interoperability is weak at times – Vendors are also often stuck on the interoperability of various storage products and systems. The fundamental issue is that vendors need to differentiate their solutions in order to compete. However, this often results in interoperability issues or worse, vendor lock-in for customers.

- Security is lacking – Normal TCP/IP and Fibre Channel do not provide serious security for data transport. FC does have LUN Masking, but this is mostly a function of the FC switch, not the storage device itself. IPsec does provide security to IP-based applications, but adds yet another layer of complexity to an already difficult solution.
- Inability of existing storage technologies to apply to new areas – Existing network storage methods do not take non-traditional applications and areas into account. An example of this is in home and personal network storage and using simple infra-red, Bluetooth or wireless LAN for small data access or transport.
- Difficulties in scaling – Existing systems scale upwards through new higher bandwidth standards. This is often slow due to the standards process. Furthermore, the scaling of capacity is difficult due to the continuous need to build and implement larger and larger disk systems that are generally not modular enough.

Based on this background, we set about designing, developing and testing a new network storage protocol that we hope will address these and other network storage issues. We would like to present some of the results from our research and development efforts that began in June 2000 in this paper.

2. Existing Solutions

Recent efforts in network storage have expanded to include development of alternatives to pure Fibre Channel as the primary method for network storage. These efforts include iSCSI, FCIP, SST and many others. Below are descriptions of a few of these efforts.

2.1. Fibre Channel over TCP/IP (FCIP)

Fibre Channel Over TCP/IP (FCIP) describes mechanisms that allow the interconnection of islands of Fibre Channel storage area networks over IP-based networks to form a unified storage area network in a single Fibre Channel fabric. FCIP relies on IP-based network services to provide the connectivity between the storage area network islands over local area networks, metropolitan area networks, or wide area networks [1]. What this means is that FCIP is designed to encapsulate Fibre Channel over a TCP/IP-based network for the purposes of connecting dispersed FC-based SANs.

2.2. iSCSI

The iSCSI Internet Draft describes a transport protocol for SCSI that operates on top of TCP [2]. iSCSI enables the use of SCSI devices over a TCP/IP-based infrastructure. Other areas considered include Naming and Discovery, Boot and Security. It is important to note that iSCSI is the only protocol currently in the process of standardisation that allows for the construction of native end-to-end Ethernet SANs [3].

2.3. Internet Fibre Channel (iFCP)

iFCP specifies an architecture and gateway-to-gateway protocol for the implementation of Fibre Channel fabric functionality on a network in which TCP/IP switching and routing elements replace Fibre Channel components. The protocol enables the attachment of existing Fibre Channel storage products to an IP network by supporting the fabric services required by such devices [4]. The purpose here seems quite clear, that is to implement Fibre Channel fabric architectures over a TCP/IP-based network, thus allowing FC devices to connect and run FC natively over a TCP/IP-based infrastructure.

2.4. Metro FCP (mFCP)

mFCP is a UDP-based implementation of the iFCP over metro- and local-scale IP networks. These networks are provisioned to have latency, reliability, and performance levels comparable to that of a Fibre Channel network. Storage devices use the Fibre Channel SCSI mapping in FCP for data transport and error recovery. mFCP leverages these existing mechanisms to facilitate high-performance interconnection of Fibre Channel- based storage devices over suitably provisioned IP networks. As in the case of iFCP, Fibre Channel frames may be transported natively over such a network without Fibre Channel switching and routing elements [5].

2.5. Internet Storage Name Service (iSNS)

iSNS provides a generic framework for the discovery and management of iSCSI and Fibre Channel (FCP) storage devices in an enterprise-scale IP storage network. iSNS is an application that stores iSCSI and FC device attributes and monitors their availability and reachability in an integrated IP storage network. Due to its role as a consolidated information repository, iSNS provides for more efficient and scalable management of storage devices in an IP network [6]. iSNS is meant to be used with iSCSI, FCIP, iFCP and such protocols for the hosts or servers to locate and use storage devices over a large network infrastructure such as the Internet.

2.6. SCSI on Scheduled Transfer Protocol (SST)

The SCSI on STP standard defines a transport protocol within the SCSI family of standards. The physical interconnects to which the SST protocol may attach are not defined within this standard, but rather, are any interconnects or other protocols on which the basic ST protocol may operate [7]. SST defines a mapping to carry SCSI traffic on top of an STP-based infrastructure.

2.7. Basic Technologies

The above technologies are built on top of a basic set of storage technologies. There are two such basic command sets today, ATA/IDE and SCSI. Based on these two command sets, other derivative technologies have been developed. See Table 1 for a pictorial representation of these technologies.

Base Command Set	**ATA/IDE**	**SCSI**
Derivative / New Developments	ATA 133 Serial ATA	SCSI-320 Universal Serial Bus IEEE 1394 "FireWire" Fibre Channel SSA
Network Storage Developments		iSCSI iFCP FCIP SST

Table 1: Storage Technologies

At this point, we turn our attention to our development efforts of the HyperSCSI protocol. Further in this paper, we will present a few ideas for thought regarding HyperSCSI and various other technologies.

3. The Approach

The first thing we decided on was to standardise on using the Small Computer Systems Interface (SCSI). It is the predominant mechanism for various storage and even non-storage devices. The question then turned quickly to how we could make SCSI "network-enabled". This gave rise to our idea of "HyperSCSI".

We found that the requirements of local network storage (SAN) and wide-area network storage (SWAN) are quite different. As such, we provided the capability to spilt HyperSCSI protocol into multiple modes of operation. Two such modes are currently being developed, one for local access, Local HyperSCSI over Ethernet (HS/eth), and the other for wide-area connectivity, Wide-Area HyperSCSI over IP (HS/IP). The basic protocol structure is essentially the same, thus allowing devices to speak local or wide-area storage seamlessly. This has allowed us to adopt IP-based networking technologies for wide-area applications where it is needed but bypassing IP entirely and putting the protocol directly onto Ethernet itself for optimum local area communications. This model also allows us to eventually develop HyperSCSI for other technologies, such as Asynchronous Transfer Mode (ATM) for high speed Telco / ISP environments and Wireless LAN for home or personal network storage.

Furthermore, since we are designing a low-level protocol, some of the intelligence or command and control functions can be passed on to higher layers or the clients to adapt and handle. This allows us to design a protocol that is lightweight and efficient, while leveraging the intelligence and capabilities of both the storage system and host machine

to mutual benefit. For example, we allow device, security and compression options as well as storage virtualisation technologies to be implemented on either the storage system, host machine or both as the needs arise. In addition, packetisation and virtualisation options of HyperSCSI allow us to implement N-channel communications technologies in order to use "scale-out" methods of bandwidth and capacity increases with fault tolerance and reliability. Figure 1 shows a Local HyperSCSI packet on Ethernet (HS/eth). A wide-area HyperSCSI (HS/IP) packet is essentially the same, but built on IP instead of directly on the Ethernet.

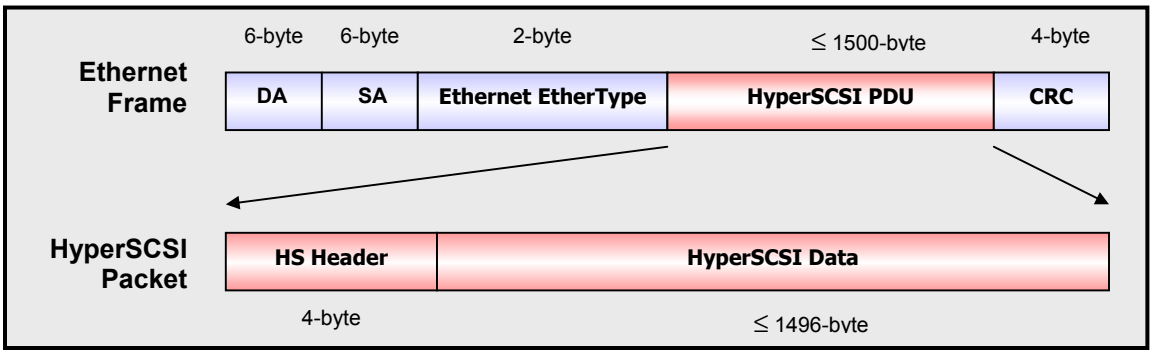


Figure 1: HyperSCSI Packet

Finally, more advanced functions and capabilities were built into the HyperSCSI protocol to support other requirements like dynamic management, dynamic flow control and in-band management capabilities. Manufacturers, system integrators and technology companies are not left out in the cold either. To enable the protocol to be interoperable, and yet be able to support vendor-specific or implementation-specific functions, a special set of dynamically negotiated device options has been designed into the protocol. These options can be negotiated at connect time and depending on the configuration of the clients and servers, be enforced, supported or ignored. Thus, HyperSCSI can provide a minimum level of connectivity for interoperability operations and while supporting advanced vendor-specific or implementation-specific functions. Our initial encryption methods demonstrate this function in action. Other possible device specific options include read-only access, removable media locking and data compression.

4. HyperSCSI Operation

The HyperSCSI protocol comprises of various packet structures. These structures are categorised by classes and then by specific types. Packets of a specific class and type may also have more than one function depending on the context of the communication. These packets are responsible for transmitting the SCSI data and commands as well as managing the connection and communication channel. Table 2 illustrates some of the packets in the HyperSCSI protocol.

HyperSCSI Packet	Description
HyperSCSI Command Block Encapsulation Class	
HCBE_REQUEST	HyperSCSI command block encapsulation request
HCBE_REPLY	HyperSCSI command block encapsulation reply
HyperSCSI Connection Control Class	
HCC_DEVICE_DISCOVERY	Client issues this packet to discover storage devices on the network
HCC_ADN_REQUEST	Authentication challenge and device operation negotiation request
HCC_ADN_REPLY	Authentication and device operation negotiation reply
HCC_DISCONNECT	Termination of HyperSCSI connection
HyperSCSI Flow Control Class	
FC_ACK_SNR	Flow control set-up and acknowledgement request
FC_ACK_REPLY	Acknowledge reply

Table 2: HyperSCSI Operations

5. Typical HyperSCSI Connection Flow Sequence

Figure 2 illustrates a typical sequence of the communication stages between a client and server using the HyperSCSI protocol. The various stages of the connection flow sequence are described below.

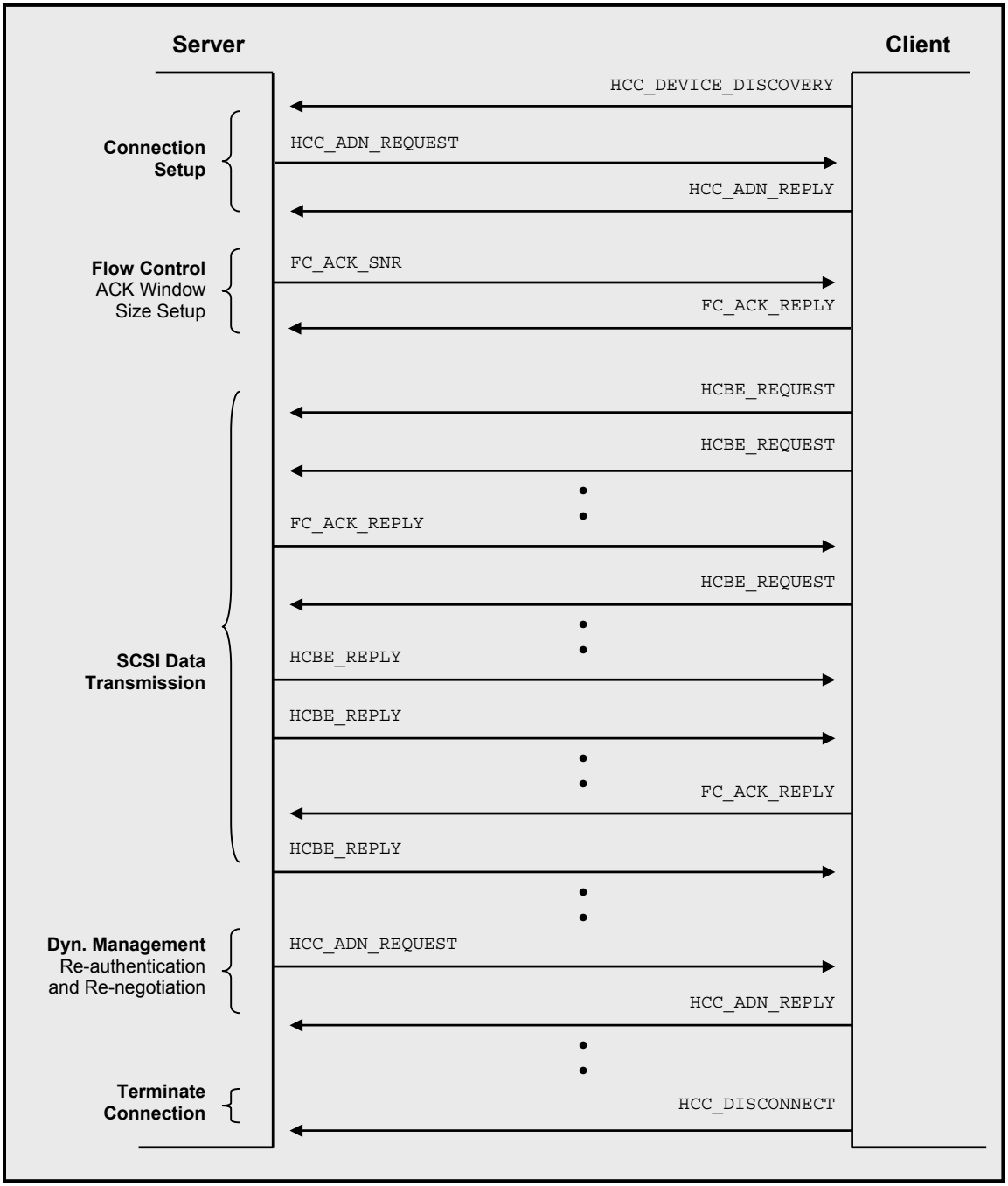


Figure 2: Typical HyperSCSI Connection Flow Sequence

5.1. Connection Setup

The HyperSCSI connection setup is a three-step handshaking procedure between a HyperSCSI client and server pair. Typically, in a storage network, the host machine (HyperSCSI client) is responsible for locating and initiating connections to storage devices (HyperSCSI servers). During this process, the HyperSCSI client issues a HCC_DEVICE_DISCOVERY via Ethernet broadcast or IP packet, to locate devices on the network. For IP-based situations, neither broadcast nor multicast methods are used. Instead, a client must specify an IP address (or DNS name) and a HCC_DEVICE_DISCOVERY packet is sent over IP directly to the server. Further information about device discovery is covered in section 6.2. Once the HyperSCSI server receives this packet, it checks the client address for authentication purposes and transmits the HCC_ADN_REQUEST packet back to the HyperSCSI client. In order for the HyperSCSI client to establish a connection with the HyperSCSI server, it must then send the correct response through a HCC_ADN_REPLY command and add the ID numbers of the devices that it has access to into its own registry. If the server successfully authenticates the HCC_ADN_REPLY, the connection is accepted and the HyperSCSI client can now send commands to the server. Within the HCC_ADN request and reply method, authentication challenges, encryption key exchanges, device specific option negotiations and other information supporting N-channel communications such as server/client IDs and network addresses are also provided and exchanged.

5.2. Flow Control and ACK Window Size Setup

An ACK mechanism has been adopted to support flow control of data between an HyperSCSI client and server pair. The ACK window size refers to the number of packets that the transmitter may continuously send before waiting for an acknowledgement. This window size must be negotiated and agreed upon before data flow can take place and is set by the requestor through an FC_ACK_SNR command. This packet is issued as a separate message and typically, the server will be the one to issue this command so that the server has the ability to balance loads or priorities across multiple clients, although this does not mean that the client may not issue one either. Once the FC_ACK_SNR has been received, the new status will be acknowledged to the requestor with an FC_ACK_REPLY. If the requestor receives the acknowledgement, it assumed that the window size is accepted and packet transmission using the new window size can begin. The ACK window size can be set based on traffic loads, or buffer capacities and can be set at start-up or changed dynamically during run time. This allows for different window sizes to be dynamically set by clients and servers to fit changing performance, reliability or QoS requirements. For example, under bad network environments, windows sizes can be reduced, while under optimum situations, window sizes can be increased for better performance. However, we are still studying algorithms for the detection of network congestion and updating of the window size during run time. The basic protocol supports this capability and we will include this portion when it is complete. Transmission windows used here are neither fixed nor sliding in nature, but rather utilises a moving window scheme similar to credit-based schemes used in Fibre Channel, but measured in windows rather than individual packets. In addition, the FC_ACK_REPLY is also used to acknowledge the correct reception of a window to the requestor and synchronises the data flow between an HyperSCSI client and server pair. In this case, it functions as an

indicator of the receiver status for normal HyperSCSI data transmission. If the transmitter does not receive the correct FC_ACK_REPLY packet within a timeout period, it will re-transmit all the packets in the window in question again. Another retransmission scheme supported is by using the FC_ACK_SNR to query the receiver's status. The transmitter can then use the FC_ACK_REPLY results to re-calculate the next packet to be transmitted. With these two schemes, re-transmits can be conducted selectively or by ACK windows, thus giving users a high level of flexibility in controlling the flow of data and commands.

5.3. HyperSCSI Data Transmission

When there is a SCSI request from the local OS SCSI upper layer of the host machine, the HyperSCSI client software is responsible for converting the OS-specific SCSI command block together with any relevant data (as in a write command) into a platform independent HyperSCSI Command Block (HCB). The client then encapsulates and fragments the HCB into one or more HCBE_REQUEST packets that it sends to the HyperSCSI server. SCSI command blocks and user data will therefore be transmitted together in the same packet. The HyperSCSI server receives the data stream, re-assembles the HyperSCSI command block and relevant user data, converts it back to an OS-specific SCSI command block and passes it to the relevant hardware for execution. When the result of this SCSI request is ready, the HyperSCSI server will send the result together with the requested data back to HyperSCSI client by issuing the HCBE_REPLY packet stream in a similar manner as the request. The HyperSCSI client reassembles the HyperSCSI command block and converts it back to a OS-specific SCSI command block before passing it on to the local OS SCSI upper layer. During this transmission, flow control mechanisms are in effect through the use of FC_ACK_REPLY commands as described in section 5.2.

5.4. Dynamic Management

During a HyperSCSI connection, the HyperSCSI server will regularly (timer-based) issue a HCC_ADN_REQUEST command for three purposes, re-authentication of clients and key-exchange for security, re-negotiation of device options (if permitted), and as a form of "keep-alive". Through this method, servers not only poll the client's status, but also check its identity. Furthermore, if HyperSCSI encryption options are turned on for data transmission, the HCC_ADN_REQUEST and HCC_ADN_REPLY uses an authenticated exchange mechanism to update and change encryption keys. This scheme also allows a device's options to be modified dynamically. For example, a device which does not have encryption enabled may turn it on during this time so that the communication will be secured from this point onwards. To enable such remote management functions, an encrypted Management Command Stream is used to transfer management commands from a client to a server or vice-versa. This MCS also allows adding or removing clients, requesting the change of device options, changing access passwords and device access permissions. The MCS is implemented within a valid HyperSCSI connection, thus only authenticated HyperSCSI clients and servers can use this in-band management mechanism.

5.5. Connection Termination

The HyperSCSI client can close a connection by sending an HCC_DISCONNECT command to the HyperSCSI server. The server will then remove this client from its connection list and close the connection. Servers do not need to acknowledge disconnect requests from clients because SCSI connections are host-target based. Unlike TCP/IP connections, which are full-duplex and can be closed by both clients and servers, SCSI connections can only be terminated (gracefully) by clients. If a server were to terminate a connection, it implies that service has been lost (or a hard disk has crashed). Servers do not keep connection information forever, and will drop relevant connections if "keep-alives" (as outlined in section 5.4) to a particular client should fail for some reason. Through the use of hashing, encryption and security methods (see section 6.3), connections are protected from denial of service attacks from hackers arbitrarily using the HCC_DISCONNECT command.

6. Feature Comparison

There are many points to consider when making comparisons of HyperSCSI features to other technologies. In the area for security for example, HyperSCSI makes use of sequence numbers, hashing, SCSI command identifiers, digital keys and other mechanisms to secure a connection, similar in some areas to IP and SCSI. A point to note has been that where possible, we have tried to adapt good ideas and mechanisms from other technologies for use in HyperSCSI. A good reference is the six manipulation functions used in any data transport protocol [8]. Thus, while differences exist, similarities will definitely show up as well in any comparison with HyperSCSI. Presented in Figure 3 are some ideas for consideration.

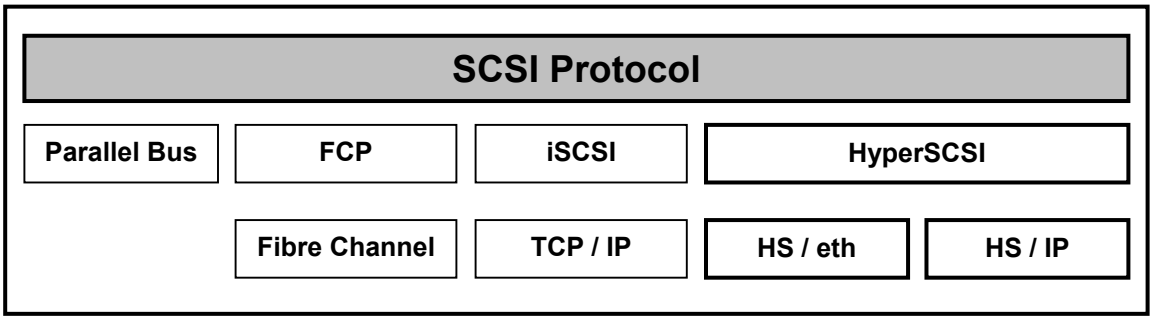


Figure 3: Protocol Stack Comparison

6.1. Storage Device Management

As it turns out, this is a key aspect of network storage that is often neglected. Proprietary enterprise management software or dedicated SAN management software from vendors or switch manufacturers is often required to properly manage the storage devices. Fibre Channel devices, switches and arrays often have an additional Ethernet port and IP address for access from the management software. HyperSCSI provides an in-band management mechanism that allows properly authenticated (and permitted) clients and servers to manage each other's settings and properties. Some device and management options can even be modified and updated dynamically during a connection.

6.2. Device Discovery Mechanisms

To identify and locate storage devices, Fibre Channel has World Wide Name (WWN) while iSCSI/FCIP/iFCP use iSNS. Such mechanisms are complex and add another hindrance to achieving ease of use and even plug-and-play networking. For this purpose, HS/eth uses standard a broadcast device discovery mechanism to dynamically locate targets on the network. If a server is configured to allow a particular client to attach, it will respond appropriately, else the discovery request is ignored. Thus the only configuration users have to be concerned about is granting permissions, rather than setting up complex name servers of some type. This is particularly useful in a plug-and-play wireless personal storage network environment. HS/IP on the other hand, leverage standard DNS mechanisms to "locate" a server across the network. We do not endorse the idea of "broadcast / multicast to find out who's out there on the Internet" as a means to locating storage resources. Storage being a key and critical resource should be managed as securely as possible, especially if it is on a public or private IP-based network. If protocols can be routed, physical security of the storage network is less assured. As such, administrators should know before hand the IP or DNS address of the client and server, configure them accordingly and not have such information "discovered" for security reasons. This also means that there is no single point of failure like having iSNS servers or requiring expensive switches with additional intelligence built-in. HyperSCSI clients will then attempt to connect to the server address given to it, and no other. The only configuration that users need to worry about in the end is granting permissions.

6.3. Security

All three TCP/IP based encapsulation methods iSCSI, iFCP and FCIP provides for and requires the use of IPsec for securing the TCP/IP connection. Certainly, this is a step forward when considering that Fibre Channel's main security mechanism is LUN masking which is implemented mostly on the switch. However, using IPsec implies securing the entire connection. This is different from the more flexible LUN masking method that FC uses to allow the user to secure individual LUNs as the case may be. HyperSCSI thus supports security options to be specified by individual devices (or LUNs) instead of at the connection level. Of course, iSCSI for example, only supports one LUN per connection, while HyperSCSI can have multiple devices in a single connection, as outlined in section 6.4. It should also be noted that like Fibre Channel, HS/eth (which does not use IP at all and is not routable) would require physical access to the network in order to hack it. HyperSCSI also allows for security to be modularised into different levels of requirements such as hashing, encryption or none at all, thereby giving even more options to secure (or not) the device and/or the connection.

6.4. Multiple Device Access

iSCSI uses one or more TCP connections to make up a single session and requires that across all connections within a session, an initiator sees only one "target image". All target identifying elements, like LUNs, are the same [9]. While this makes sense in a pure SCSI environment, where a single host bus adapter would see a single target to have one "target image", this may not be true in a network storage environment where usually disk arrays of one or more targets may be "exported" to the initiator. HyperSCSI on the other

hand allows a single connection to have access to as many SCSI devices (or LUNs) as supported by both the initiator and target. This single connection can then be established as a virtual channel over multiple physical links to form a redundant trunk. Devices that may require multiple LUN access includes optical jukeboxes and tape libraries.

6.5. Optimising Performance

One of the most controversial aspects of performance for network storage are the overheads of TCP/IP. Industry analysts have noted that the TCP/IP stack is very CPU intensive and without complex optimisation techniques like hardware accelerators, interrupt coalescing, checksum offloading, and so on [10], the only practical application for iSCSI is to extend current Fibre Channel SAN-to-LAN connectivity into the realm of SAN-to-MAN/WAN connectivity [11]. If every implementation were to require TCP/IP implemented in hardware, it would be no different than requiring all devices to have Fibre Channel hardware built-in. HyperSCSI can bypass TCP/IP entirely to build a storage network similar to (and capable of replacing) Fibre Channel architectures, but using plain old Ethernet instead. For wide area implementations, HyperSCSI does in fact also support the use of IP-based infrastructure for building Storage Wide-Area Networks through HS/IP, a strategy which is no different from Fibre Channel. It should also be noted that while HS/eth reduces overheads partly by eliminating certain checksums (ie. header checksum), IPv6 also does away with the header checksum. IPv6 designers felt that the risk was acceptable given that data link and transport layers check for errors [12]. Another key point of HyperSCSI is its reliance on state tables so that information about a connection does not have to be retransmitted over and over again. Such information includes SCSI host/target information, device options and HyperSCSI sequence numbers. This is also similar in idea to STP's architecture of setting up the receiving buffer and related information before transmitting data [13]. This is also a security benefit since the capture of a single packet is unlikely to reveal much information about the connection itself. For HS/IP, only one IP port is required, since each client can access multiple devices through a single connection, unlike iSCSI (see section 6.4).

6.6. Flow Control Issues

Fibre Channel is often touted as the best solution for network storage due to its high speed packetised but dedicated channel for storage. iSCSI on the other hand relies upon TCP/IP for flow control and packet transmission and can leverage TCP/IP's sliding windows as a counter to the idea of packetisation being less efficient compared to dedicated channels. To provide the best of both worlds, HyperSCSI adopts a moving window mechanism but makes the window size dynamic. A balance is provided in that the window size does not fluctuate like TCP/IP's sliding windows, but can and does change dynamically in the middle of a connection. Since this window size is dynamically controlled by clients and servers, algorithms for determining the window size can be adopted to find the optimal window size during run-time, thus adapting to network congestion. This is particularly evident in HS/eth implementations. HS/IP of course leverages standard IP-based methods for flow-control issues. In addition, retransmission can be implemented either using a selected retransmission scheme or a simpler window retransmit scheme. This can be decided based on the implementation environment, thus giving users a wide degree of flexibility and performance tuning options.

6.7. Simplicity, Interoperability and Diversity

HyperSCSI is designed from the ground up to be simpler for users to implement and yet capable of achieving interoperability without sacrificing diversity. For this purpose, negotiable device options allow for vendor-specific or implementation-specific features to be supported. If different vendor devices with different supported device options were to try to connect to each other, the worst case is expected to be a basic connection with no additional features or functions. When used in conjunction with the varied SCSI-3 command set and the Management Command Stream, this becomes quite a powerful value-added option for vendors and users alike.

7. Development Progress

We have implemented and tested HyperSCSI under various conditions over Fast Ethernet and Gigabit Ethernet. The results so far have proven to be most encouraging. Today, HyperSCSI on Gigabit Ethernet achieves a quick 96% of the local physical disk performance compared to iSCSI's 82% for block level access. The results are even better when considering file system level tests. Using a straightforward file copy test, HyperSCSI can reach 88% of the local physical disk performance, iSCSI managed 43% while NFS only succeeded to match 39%. Not only that, it can be seen that HyperSCSI provides a more reliable and predictable performance level similar to that of the local physical disk than iSCSI or NFS and is less dependent on caching to achieve performance. One might wonder why iSCSI performance is not as good as expected. Seeing how iSCSI performance seems to closely track NFS performance, we hypothesise that the TCP/IP overhead is the differentiating factor between iSCSI and HyperSCSI performance. The following charts highlight some of the performance measurements that we have conducted.

The results illustrated in Figure 4 represent results from five different tests, two of which were raw block level reads (hdparm and dd) and the other three represent data access above the file system level. These tests were done on the same hardware and the same OS for all three technologies and both the client and server. We used two AMD Athlon 1.2GHz SMP machines with 32-bit 33MHz PCI busses, 266MHz 256MB DDR RAM running RedHat Linux 7.1 using the standard Linux kernel version 2.4.16, one of which was the client and the other was the server. Both machines had 3Com 3C985B-SX Gigabit Ethernet NICs, connected over a cross connect fibre-optic cable with jumbo frames, and the server used an Adaptec 39160 U160 SCSI controller. The server exported 8 IBM UltraStar U160 9.1GB 10k RPM drives configured in RAID 0. For the tests using a file system, Linux Ext2 was used as the file system. We used NFS version 2 over UDP from the RedHat Linux RPM version 0.3.1-5. The iSCSI version we used was version 6 from Intel, while the HyperSCSI version was 110-011226. The destination for the cp test was /dev/null while the Iozone version used was 3.71. We would like to draw attention, not to the absolute numbers of MB/s, but rather to the performance comparisons between iSCSI, NFS and HyperSCSI.

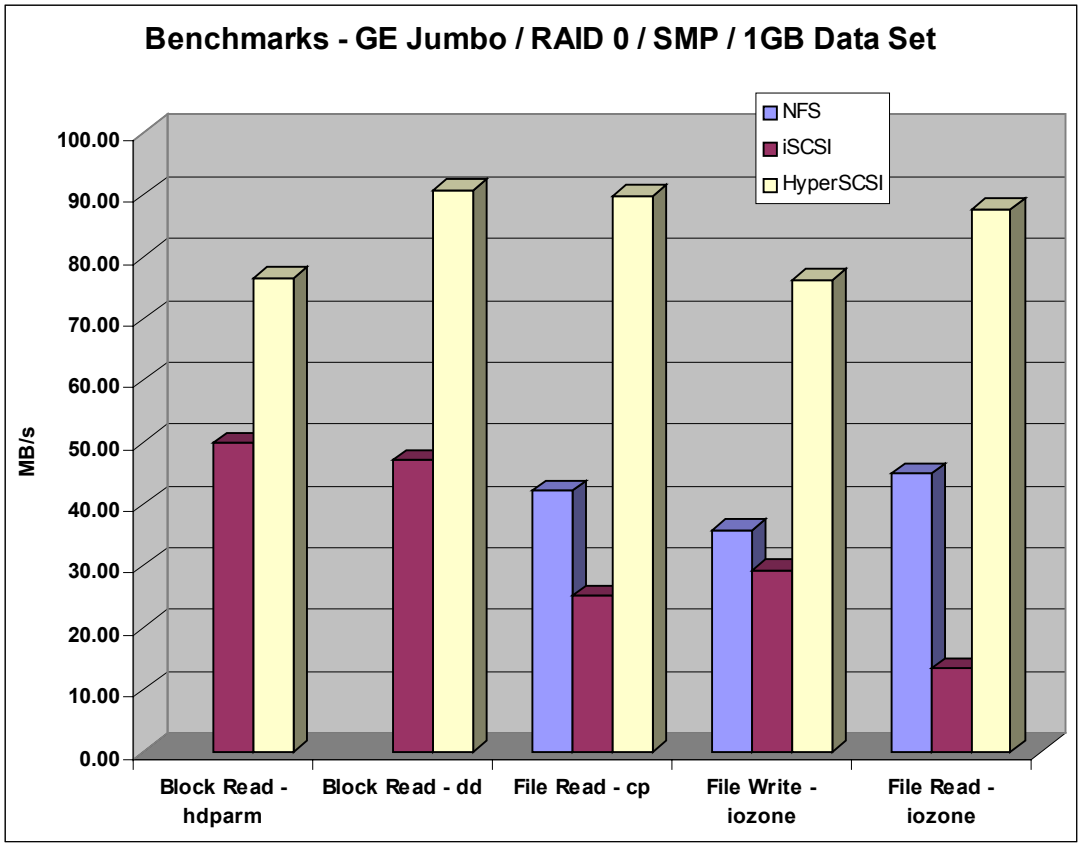


Figure 4: HyperSCSI Block and File Access Performance Comparison

Feature-wise, the HyperSCSI reference implementation already supports standard SCSI hard drives, IDE hard drives, software RAID / virtualised drives, optical disks (like DVDROM and CDRW), USB devices (like Iomega Zip Disk) and SCSI tape drives (like HP DAT40). We have even successfully used HyperSCSI is to write CDs remotely over our own live corporate LAN. File systems like Microsoft's FAT16/FAT32, SGI's XFS, IBM's JFS and Linux Ext2/Ext3 have all been successfully tested on HyperSCSI drives. HyperSCSI clients and servers have been successfully implemented on Linux, while client versions on Windows 2000 and Solaris 8 is currently in development. Encryption schemes that have already been implemented include 64-bit Blowfish and 128-bit Rijndael. HyperSCSI has been assigned its own IEEE Ethertype Number, and will soon receive a registered IP port for HS/IP implementations.

Areas that are currently under development (at the time of writing of this document) include aspects of the Management Command Stream, the Transmission Pause / Resume, various hashing and security related options, HS/IP implementation and Windows 2000 and Solaris 8 versions of the Linux client. With continued optimisations and bug-fixes of the reference implementation, we expect raw block data read speeds for a RAID0 subsystem of 8 drives on normal frame Gigabit Ethernet to exceed 100MB/s in early

2002. Another effort underway is the testing of several N-channel communications schemes for HyperSCSI. A Peer Round-Robin scheme is likely to be used in the final implementation. The documentation of HyperSCSI specifications is also critical in order to allow other organisations to adapt and build their own HyperSCSI solutions. Currently there are three documents in the HyperSCSI specifications, HyperSCSI Protocol Specifications, HyperSCSI Security Specifications and HyperSCSI Management Command Stream Specifications. A Quick Reference Manual, Reference Implementation Source Code Documentation, and various introductory documents like this one will also be provided. These documents will be available on our website when completed.

8. HyperSCSI Applications and Conclusion

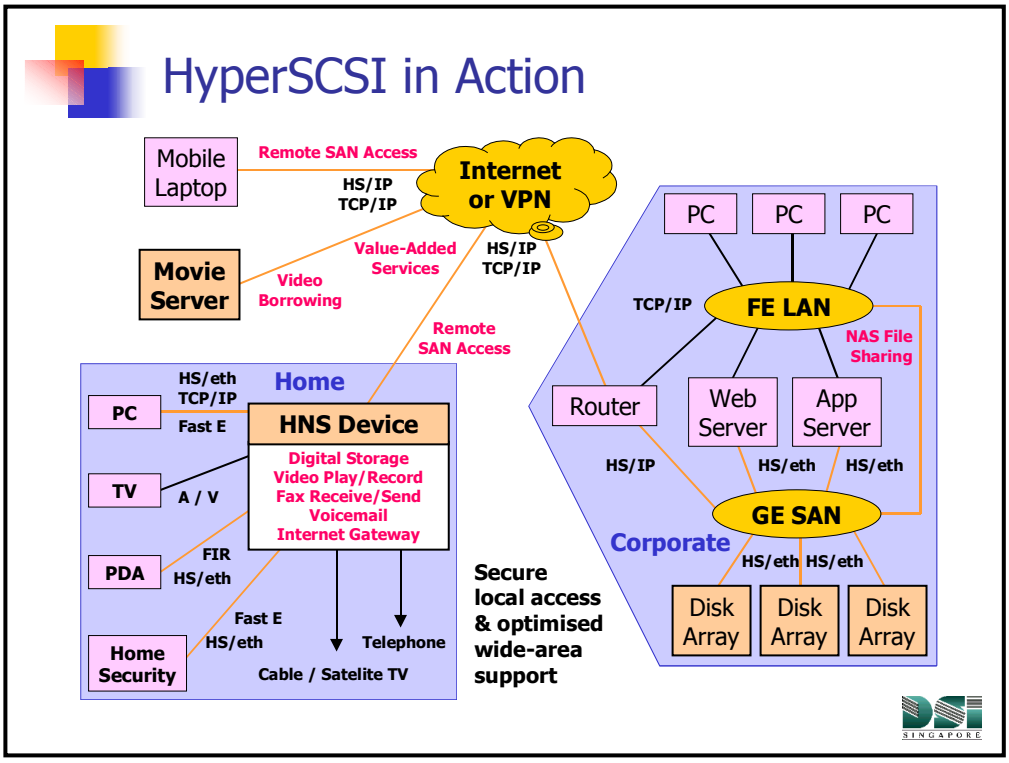


Figure 5: HyperSCSI in Action

We believe that HyperSCSI provides an opportunity to address various concerns and open up new possibilities for network storage. The Local HS/eth protocol allows the construction of high-speed Ethernet based SANs while the use of Wide-Area HS/IP permits mobile devices like laptops to access the corporate SAN directly (bypassing servers if need be). Storage devices can support SAN or NAS or both access methods simultaneously through the use of a single network interface. Home devices will also be able to access storage directly with simple plug-and-play methods over Fast Ethernet or Wireless LAN using HyperSCSI's device discovery schemes. HyperSCSI has also been designed with the future in mind. It supports more than 32,000 different device options that will allow vendors to introduce a wide variety of vendor-specific capabilities and technologies, without sacrificing interoperability. The protocol also allows each single

HyperSCSI connection to handle 64 simultaneous in-transit SCSI commands, each with SCSI command block sizes up to 512KB. These SCSI command block sizes can be further increased six-fold by using Gigabit Ethernet jumbo frames, thus providing an even higher level of performance.

In conclusion, we believe that HyperSCSI is a relatively simple technology that can provide users with performance, security, scalability and flexibility, thus making it a viable alternative solution for network storage applications.

For more information on HyperSCSI, please visit our website at http://nst.dsi.nus.edu.sg/mcsa/

9. Acknowledgements

The authors would like to thank various people who have helped tremendously in the writing of this paper. First and foremost of course is Rod Van Meter who provided valuable feedback and reviews of our work, and in so doing helped us produce a better paper. In addition, various people have contributed tremendously in the development and testing of HyperSCSI, including Alvin Koy, Ng Tiong King, Vincent Leo, Don Lee, Premalatha Naidu, Wang Hai Chen and of course, Wei Ming Long, who carried out a lot of work on the testing environment. Thanks also to Jimmy Jiang, Lalitha Ekambaram and Yeo Heng Ngi, who have contributed much to the development of HyperSCSI on Win2K. This has been a team effort and our thanks and appreciation goes out to all that have helped in one way or another.

References

[1] IPS Working Group Internet Engineering Task Force, "Fibre Channel over TCP/IP (FCIP) Internet Draft", November 2001

[2] IPS Working Group Internet Engineering Task Force, "iSCSI Internet Draft", November 2001

[3] Brent Ross, Adaptec Inc, "IP Storage and iSCSI – The SAN Fabric of Choice", Proceedings of Storage Area Networks Conference 2001

[4] IPS Working Group Internet Engineering Task Force, "iFCP – A Protocol for Internet Fibre Channel Storage Networking Internet Draft", November 2001

[5] IPS Working Group Internet Engineering Task Force, "mFCP – Metro FCP Protocol for IP Networking Internet Draft", May 2001

[6] IPS Working Group Internet Engineering Task Force, "Internet Storage Name Service Internet Draft", November 2001

[7] American National Standard of Accredited Standards Committee ANSI NCITS T11.1 Technical Committee, "Information Technology – SCSI on Scheduled Transfer Protocol (SST) Work Draft", July 2001

[8] David D Clark and David L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", ACM SGICOM 1990 Symposium

[9] IPS Working Group Internet Engineering Task Force, "iSCSI Internet Draft", November 2001

[10] Jeffrey S. Chase, Andrew J. Gallatin and Kenneth G. Yocum, "End System Optimisations for High-Speed TCP", IEEE Communications Magazine, April 2001

[11] Roy Levine, "IP-based Storage: Benefits and Challenges", Infostor, March 2001

[12] Andrew Conry-Murray, "Internet Protocol Version 6", Network Computing, December 2001

[13] American National Standard of Accredited Standards Committee ANSI NCITS T11.1 Technical Committee, "Information Technology – Scheduled Transfer Protocol Revision 4", October 2000

[14] Network Working Group Internet Engineering Task Force (IETF), "Internet Protocol Version 6 (IPv6) Specification RFC 2460", December 1998

[15] American National Standard of Accredited Standards Committee ANSI NCITS T10 Technical Committee, "SCSI-3 Architecture Model (SAM)", 1996, Revised 2001

[16] American National Standard of Accredited Standards Committee ANSI NCITS T11 Technical Committee, "Fibre Channel Protocol (FCP)", 1996, Revised 2001

[17] Information Sciences Institute University of Southern California, "Internet Protocol DARPA Internet Program Protocol Specification RFC 791", September 1981

[18] Information Sciences Institute University of Southern California, "Transmission Control Protocol DARPA Internet Program Protocol Specification RFC 793", September 1981

Point-in-Time Copy: Yesterday, Today and Tomorrow

Alain Azagury, Michael E. Factor and Julian Satran
IBM Research Lab in Haifa
MATAM, Haifa 31905, Israel
{azagury, factor, satran}@il.ibm.com
Phone: +972 4 829-6211, Fax: +972 4 829-6116
William Micka
IBM Storage Systems Group
9000 S RITA ROAD, Tucson, AZ, USA
micka@us.ibm.com
Phone: +1 520 799-4132

1. Introduction

Making copies of large sets of data is a common activity. These copies can provide a consistent image for a backup, a checkpoint for restoring the state of an application, a source for data mining, real data to test a new version of an application, and so on. One characteristic all of these uses have in common is that it is important that the copy appear to occur atomically, *i.e.,* any updates to the data source being copied either occur before or after the copy. In this work, we examine the history, the state-of-the art, and possible future of mechanisms for copying large quantities of data via storage subsystem facilities for providing *point-in-time* (PiT) copies.

The Storage Networking Industry Association (SNIA) defines a *point-in-time copy* as:

> A fully usable copy of a defined collection of data that contains an image of the data as it appeared at a single point-in-time. The copy is considered to have logically occurred at that point-in-time, but implementations may perform part or all of the copy at other times (e.g., via database log replay or rollback) as long as the result is a consistent copy of the data as it appeared at that point-in-time. Implementations may restrict point-in-time copies to be read-only or may permit subsequent writes to the copy. Three important classes of point-in-time copies are split mirror, changed block, and concurrent. Pointer remapping and copy on write are implementation techniques often used for the latter two classes. *cf.* snapshot [1]

As hinted at by the above definition a range of point-in-time copy facilities exist. Some of these facilities operate at the logical level of the file system [2][3] and some operate at the physical level of the disk storage subsystem [2][4][5][6]. We focus on copy facilities provided by disk storage subsystems.

Before the invention of point-in-time copy facilities, to create a consistent copy of the data, the application had to be stopped while the data was physically copied. For large data sets, this could easily involve a stoppage of several hours; this overhead meant that there were practical limits on making copies. Today's point-in-time copy facilities allow a copy to be created with almost no impact on the application; in other words, other than perhaps a very brief period of seconds or minutes while the copy is established, the application can continue running.

This paper describes the functionality of a point-in-time copy facility and describes both the benefits and drawbacks of providing this facility on the storage subsystem. While there are other benefits, the biggest benefit of providing this facility on the storage subsystem is performance; we do not needlessly add load to the storage network or host as part of making the copy. The biggest drawback is that the storage subsystem in today's world is only aware of data at the level of logical units and blocks;[i] this makes it hard to meaningfully perform copies at a granularity of less than an entire logical unit.

After defining the concept of point-in-time copies, we briefly survey several existing approaches including EMC's TimeFinder[4], IBM [7] and StorageTek's [8] virtual array solutions, and several file system based approaches. Although the focus of this paper is on point-in-time copy solutions for block controllers, we also describe file system snapshots, in particular Network Appliance's snapshot feature [9]. We then describe the FlashCopy facility of IBM's Enterprise Storage Subsystem (ESS) [6] which was developed in our labs; we present performance results showing that this facility allows copying arbitrary amounts of data in almost zero time.

We then describe one possible future for point-in-time copy facilities. We see two main future thrusts for point-in-time copy facilities. This first is improved performance; while today's facilities can make a copy in almost zero time, even this is sometimes too much time. The second is a melting of the division between the organized logical view of data implemented by a file system and the physical view as seen by today's disk subsystems [10]. In particular we believe that the arrival of object based storage *e.g.,* [11], will provide a critical enabler for allowing a disk subsystem to provide a physical point-in-time copy of logically meaningful data.

The rest of this paper is organized as follows. The next section provides a background on point-in-time copy facilities, describing the different approaches to implementing these facilities and the tradeoffs between point-in-time copies at the file system level and at the storage subsystem level. Section 3 describes several existing facilities for point-in-time copy and Section 4 describes the FlashCopy facility of IBM's ESS, showing how this facility allows copying almost arbitrary amounts of data in nearly zero time. In Section 5, we describe one possible course of development for point-in-time copy solutions prior to concluding.

2. Background

As we stated in the introduction, a point-in-time copy may be made for many reasons. While backup is probably the most common reason, checkpointing, data mining, testing

and other reasons also exist. In all cases, prior to making the copy, applications accessing the data must purge any caches they have; many middleware applications, such as databases, provide mechanisms to ensure that the underlying storage subsystem or file system has a consistent copy of the data without stopping the application. In addition, for copy facilities provided by the storage subsystem, the file system must ensure that it has written a consistent image of the data to the storage subsystem.

Obviously if the data to be copied involves multiple entities, *e.g.,* multiple logical units or multiple file systems, this quiescing of the application must occur atomically for all of the entities. Only after all of the copies have been made, is the application again allowed to modify the underlying data. This means that application access to the data is limited for the duration of time it takes to execute the copy.

Prior to the development of point-in-time copy solutions, the only way to make a copy of a data set was to allocate space and physically copy the to data. To ensure consistency, the application was not allowed to access the data while the copy was being executed. Since the time required to execute a physical copy is a function of the size of the data, this could easily lead to an application being unavailable for an extended period of time. This time overhead, as well as the need to fully allocate the space required for the target, limited the use of copies; one would not copy an entire volume of data every hour for purposes of checkpointing an applications state.

In none of today's popular facilities, however, does the point-in-time copy command execute a physical copy of the data. Instead the data is either copied prior to the execution of the command or some form of copy-on-write like facility is used. Not only does this reduce system overhead, but also it enables copies to be used in the range of applications listed above.

As stated above, there are different classes of implementations of point-in-time copy. In a *split mirror* a mirror of the data is constructed prior to the point-in-time copy. After a complete mirror of the data to be copied exists, the point-in-time copy is made by "splitting" the mirror at the instance in time of the copy. The biggest benefit of a split mirror solution is that the point-in-time copy executes very quickly; there is no work required in order to create tables or mark data as copy on write. On the other hand, split mirror suffers from a significant drawback in terms of advanced planning. One cannot create a split mirror at any time one wants; rather, it is necessary to plan ahead and create the mirror in advance of splitting. Since the mirror requires a complete physical copy of the data, the set up for creating a split mirror must begin significantly prior to the actual point-in-time copy. A second drawback of a split mirror solution since it is based on physical mirror copy is that it inherently requires that the space allocated for the target of the copy be equal to the space used by the source. Finally, the overall storage system performance is affected by the requirement to continuously mirror the changes until the administrator decides to split the mirror.

One variant of a split mirror solution allows the mirror to be resynchronized with the source. When this is done, only the records of the source, which have changed since the mirror was split, are copied to the source. This allows a true mirror to be created much

faster than if the entire data set needed to be physically copied to the mirror. This variant does, however, require work to create data structures to track which records in the source have been modified.

A second class of implementations is *changed block*. A changed block implementation shares the physical copy of the data between the source and the target until the data is written; this sharing can be at the level of a sector, a track, or conceivably some other granularity (we refer to this unit as a *record* below). To allow the data to be shared some form of table is used to determine where the actual copy of the data exists. When the source and target are accessed this table is used to determine from where the data is to be retrieved.

This table can be the directory that exists in virtual arrays such as log structured arrays [12] or it can be some other mechanism that is used only for purposes of supporting a point-in-time copy, such as a copy-on-write bitmap that tells whether or not a given record has been copied. A changed block approach is easy to implement on completely virtual systems, or other mechanisms, which use indirection for all accesses; however, it is also possible to implement a changed block approach in more conventional systems.

When the data is written a changed block implementation will either manipulate pointers in a directory or copy the written data. In either case, after the update the source and the target no longer share a physical copy of the given record.

A changed block implementation requires setting up the table to keep track of what records have been copied when the point-in-time copy is made; this obviously takes time that is linear in the size of the data to be copied. However, since these tables can be no more than a copy-on-write bitmap, this can be done very efficiently. One big benefit of changed block implementations over split mirror implementations is that no advanced set up is required prior to executing a point-in-time copy. Another feature of a changed block implementation is that the amount of space required is a function only of the amount of data modified.

A *concurrent* point-in-time copy is similar to a changed block implementation with one significant difference. A concurrent implementation always physically copies the data. Like a changed block solution, however, when the point-in-time copy is executed, no data is physically copied. Instead, the concurrent solution sets up a table to keep track of which data has been physically copied. It then physically copies the data in the background, using the table to synchronously copy any records that are about to be modified.

One other axis on which point-in-time copy solutions can be differentiated is whether or not the target of the copy is a first class citizen, *i.e.,* can the target be freely accessed or are there limitations on the way it is used, *e.g.,* no updates, only sequential reads, etc.

As discussed in the introduction point-in-time copies can be made either at the file system level or at the storage subsystem level. The biggest benefit of performing the copy at the storage subsystem level is that it can reduce the load on the server and on the

storage network (assuming one is being used). When the copy is made at the level of the file system, all of the computation of the copy must be made on the file server; in addition, whenever physical copies are required, the data must be transferred up through the storage subsystem, over the storage network to the server and then back down the same path. If the copy is made by the storage subsystem, we can totally avoid the overhead on the storage network and on the host.

3. Point-in-Time Copy Today

Research on storage point-in-time copy techniques is extremely scarce. Since one of the major uses of point-in-time copy is as a building block for efficient backup, the literature on backup techniques covers partially this topic [13]. In this section, we review some of the major point-in-time solutions available in the market. In addition, while we focus on disk storage subsystems, we describe two point-in-time copy techniques at the level of the file system.

3.1 Split Mirror Solutions

EMC's TimeFinder [2][4] and Hitachi's ShadowImage [5] are two examples of *split mirror* implementations. We describe TimeFinder's major characteristics. TimeFinder allows creating mirror images of standard devices. These mirrored images, called Business Continuance Volumes (BCVs), may be later *split* and accessed independently. BCV images are created in the background and several copies of a standard device may be created. BCVs can go through the following stages:

- *Establish* – a new BCV device is established and, initially, contains no data.
- *Isynch* – the point-in-time where the BCV pair is synchronized with the standard device.
- *Split* – makes the BCV volume available to the host. The content of the BCV volume is a point-in-time copy of the standard device at the time the split command was issued.
- *Re-establish* – The volume is re-assigned as a mirror of the standard device. The BCV volume is refreshed with any updates made to the standard device, and any updates to the BCV after the split are discarded.
- *Restore* – Copies the contents of the BCV back to the standard device.
- *Incremental restore* – Discards all the changes made to the standard device since the split occurred and copies updates made to the BCV device to the standard device.

The latest version of TimeFinder [14] introduces *changed block* capabilities: a new *instant split* operation allows BCVs to become immediately available to the hosts. This is achieved by copying tracks before they are modified in the standard device.

3.2 Log Structured Changed Block Solutions

IBM's RAMAC Virtual Array (RVA) [15][7] and StorageTek's Shared Virtual Array [8] are major examples of changed block solutions that leverage the log structure data structures for their point-in-time copy implementation. IBM's RVA represents a volume using a set of tables that eventually point to the set of tracks that comprise the volume. RVA also maintains a reference count for each track containing physical data. A snapshot operation from a source to a target volume requires (1) decreasing the reference count of the target tracks, (2) copying the "track" table from the source to the target and (3) increasing the reference count of the source volume tracks. RVA's snapshot is both efficient in time – requiring only to copy the track table of the source and updating the track reference counts – and efficient in space – since no copy of the user data is required.

3.3 File System Solutions

Many UNIX-like file systems have leveraged their *inode*, pointer-based data structures to implement efficient snapshot capabilities. The Andrew File System [3] implements a Clone operation that creates a frozen copy-on-write snapshot. Snapshots are read-only and are traditionally used for backup purposes, to allow backing up a consistent point-in-time snapshot, with minimal disruption of the activity on the production file system. In addition, snapshots can be used for easy restore of deleted or corrupted files.

Network Appliance's filer [2] also implements a copy-on-write-based snapshot facility [9] that creates on-line, read-only copies of the entire file system. It currently allows administrator to create up to twenty snapshots of a file system. In order to support snapshots, the free block data structure is extended to mark to which snapshots the block belongs. A block might be returned to the "free pool" only after each bit, for each snapshot is zero. Snapshot are created under the "`~snapshot`" directory. Users may retrieve files from previous snapshots, and restore them using standard file system "copy" commands.

Network Appliance has integrated its snapshot features with a SnapMirror/SnapRestore capability. SnapMirror allows automated, consistent replication of file systems to remote sites. It creates periodically a snapshot of the file system and then transfers the modified blocks to the remote site. After a baseline transfer is complete, Snapmirror leverages the snapshot bitmaps to identify which blocks need to be transferred to the remote site. SnapRestore allows restoring a mirrored snapshot to the primary.

File system snapshots are very efficient operations, since they only require keeping copies of modified or deleted files. However, since, not only the data, but also the metadata is read-only, one cannot modify metadata attributes of files in snapshots. For example, revoking access to a file from a user does not prevent him from accessing (earlier versions of) the file in previous snapshots. In addition, when a copy is required, the data must be transferred from the storage subsystem to the file system and back to the storage subsystem.

4. ESS's FlashCopy Today

FlashCopy is an ESS Copy Services function, developed in our labs, which is a *concurrent* class *point-in-time* copy operation. It utilizes *copy-on-write* bitmap techniques to maintain knowledge of which blocks of data have been a modified after the *point-in-time* copy was created. Real storage equal in size to the source data is required on the target volume. When a block of data on the source volume is modified, the previous version of that data is copied to the target volume before the new modification overwrites it. An optional background copy task may be initiated to perform the physical copy of the entire source volume to the target volume.

FlashCopy, unlike a split mirror technique, provides instant availability for read and write data on both the source and target volumes as soon as the invocation of the operation is complete. It utilizes the ESS cache and fast write functionality to mask any performance affects related to the *point-in-time* copy which may be activated for a given volume. FlashCopy operates on volumes for zSeries hosts and for volumes attached to open systems hosts. When invoked from a zSeries, the host program can specify that only a portion of the volume be copied. This is called a *sparse volume*. If portions of the volume are not allocated or are catalogs or volume table of contents, these can be excluded from the copy to the target and managed by the host software. An open systems volume is copied in its entirety.

The most important performance metric related to the creation of the *point-in-time* copy is the elapsed time required for the invocation of the copy on one pair or multiple pairs of volumes. During invocation, the application must maintain a consistent image of the data across all volumes used for the application. The amount of time required can be considered an application impact and must be minimized by the design of the copy function.

When a FlashCopy is initiated, the source and target are entered into a relationship using a bitmap table which reflects the location of the point-in-time data - either on the source volume or on the target volume. While the relationship table is being created within the ESS control unit, the two volumes are made unavailable to all customer access. The time for this operation can vary considerably with the method of FlashCopy initiation. The zSeries program DFSMSdss [15] performs various steps prior to the relationship creation period which elongates the initiation. DFSMSdss must read the Volume Table of Contents (VTOC), perform RACF security verifications, and then reserve the volumes involved for data integrity purposes. Given this task overhead, the FlashCopy initiation can take approximately 6 seconds for a 3 gigabyte volume. By contrast, the TSO FlashCopy function and the ESS Specialist Command Line Invocation does not include reading or verification steps and can take less than 2 seconds for the same relationship. By performing the invocation for many volumes in parallel, the time to complete the set of relationships is much better than the summation of individual invocations.

# of Flash Copies	dss small VTOC	dss large VTOC	TSO invoked
1	6 sec	8 sec	1.2 sec
256	48 sec	66 sec	18 sec

As can be seen from the table, the invocation time is a function of the number of volumes in the total data collection and the amount of information on the volumes as reflected in the VTOC.

Another important performance measurement is the effect on application response time and the number of I/O operations that can be executed per second while a FlashCopy relationship exists for a volume pair or number of volume pairs. Measurements were made using 256 FlashCopy pairs while running a cache standard workload which show less than 3% reduction in the I/O rate when the workload volumes are in a relationship with the *no background copy* option selected. With the *background copy* option selected, the rate reduction is about 7%.

The change to the workload response time is negligible when the *no background copy* is specified. There is negligible response time increase when the *background copy* is specified for 32 volumes or less in one control unit. With 256 volumes in *background copy* mode, the response time rises doubles until the number of background copy tasks is reduced by completing the copy for a pair of volumes.

5. The Future of Point-in-Time Copy

The world of data copies has improved significantly since the invention of the first facilities that allowed a logical copy without requiring a physical copy. However, there is still room for improvement. In the small, the improvements include improving the performance of today's solutions to reduce even further the impact on the application for creating a copy. In addition, it should be possible to provide greater flexibility in the facilities provided by storage subsystems, allowing a greater degree of knowledge of the logical structure of the data to flow down to the physical layer.

In the large, the greatest improvement may come from new storage architectures such as object based storage. With an object based storage and the appropriate file system architecture, it should be possible to completely bridge the gap between the logical structure as seen by the file system and the physical structure provided by the storage subsystem.

5.1 Improving Today's Point-in-time Copy

As fast as the execution of a point-in-time copy may be, until it is instantaneous, it will never be fast enough. This is because as described in Section 2, while the command for the point-in-time copy is executing, it may be necessary to limit application access to the data being copied.

There are several aspects to improving the performance of today's point-in-time copy solutions. First, it is important to speed up the time required to ensure that the component performing the copy has a copy of the data that is consistent with the application's view of the data. This includes ensuring that all data that is in cache has been written through to the appropriate level of the system or at the very least knowing what data needs to be retrieved from a cache.

Second, the data structures used to manage the copy need to be set up quickly. To some degree this is a problem that is inherently linear in the size of the data to be "copied"; for instance, a table recording which data has been copied must be a size that is the same order of magnitude as the size of the data. However, even here, by intelligently preparing the data structures it may be possible to hide some or most of the overhead from the application.

In addition to improving performance, we believe that point-in-time copy solutions will evolve to have more flexibility in terms of the allowing knowledge of a file system's logical structure to flow down to the storage subsystem. To a degree this exists today for mainframe data with the support for making point-in-time copies of individual data sets [6][16]. However, more work is required to provide this same facility for partitions in a way that is not tied to a particular logical volume manager or file system.

5.2 Point-in-time Copy and Object Based Storage

Object Based Storage (*e.g.*, [11]) provides the client (storage consumer) with a storage abstraction closer to the client's view of the data than the conventional device view. In conventional storage devices the client perceives a device as a collection of storage blocks (usually fixed length). Organizing this primitive storage into entities significant to applications and managing all storage resources is delegated to client software (operating system), sometimes in conjunction with a third party (a file server). Only through client and/or file server software is the client able to attach significance to data. This classical structure has two main disadvantages:

- it is hard to scale to large volumes of data and large throughput since data servers quickly become bottlenecks
- data management at storage level has no relation to content

Widely discussed in academia and now starting to happen in industrial laboratories, a new form of storage access - Object Based Storage - changes the way storage is accessed and managed.

Object Based Storage (OBS) relegates space management to the storage subsystem. Instead of perceiving a volume as an amorphous collection of equally sized storage blocks the storage client perceives now a volume as a collection of variable length (possibly sparsely populated) objects and the mapping of those objects to device-blocks is delegated to the storage controller.

Client access to data is based on an object-handle (capability) established by a management component in the network. Access to data is protected through the capability and unmediated.

In addition to enable building highly scalable storage subsystems (as the access does not have to go through a data server) Object Based Storage make access units (objects) "visible" and manageable at storage subsystem level. The previously discussed copy functions can now be performed based on policies pertinent to specific objects or classes of objects.

In addition since the storage subsystem has complete control over device block location information and internal object structure, management functions, such as point-in-time copy or incremental point-in-time copy, can be made with minimal space (and time) overhead and encompass any set of objects (not necessarily a volume or a large portion of a volume). And although the examples that follow involve files it can easily be observed that they might as well refer to database tables or any other type of storage object.

5.2.1 Point-in-time copy for a set of files

Point-in-time copy for a set of files in a file-system built using Object Based Storage involves the following steps on a client/administrative system:

1. Lock the set of files
2. Copy the directory entries for the set of files
3. Request a point-in-time copy for the set of objects containing the files data from the storage subsystem to be performed asynchronously
4. Release the locks
5. Wait for the point-in-time copy command to end

The storage subsystem will do the following:

1. Mark all the involved objects (their control structures) as copy-on-write
2. Return to the host an indication of "successful request"
3. Perform the request while accepting read/write operations from the host

It is easy to observe that given enough free space to accommodate host write operations during the point-in-time copy generation, any number of point-in-time operations can be performed simultaneously.

To perform such a point-in-time copy for a set of files using today's mechanisms, would require that we give the control unit detailed knowledge of the way the file system lays

out files. Since on disk layout differs between file systems, separate implementations would be required for each file system supported.

6. Conclusions

We have described the current state of the art of point-in-time copy operations, focusing on the FlashCopy facility of IBM's ESS developed in our labs. Using FlashCopy as an example, we have shown how today's point-in-time copy facilities can perform a semantic copy of large quantities of data in essentially zero time.

While performance of today's copy is orders of magnitude superior to the time required to make a physical copy, there is still some room to improve performance. More significantly we see that the future melding of block based and file based storage, promised by facilities such as object based storage, will provide an opportunity for storage subsystems to provide point-in-time copy for entities that are meaningful to the end user, *e.g.,* files, and not just entire or large portions of logical units.

Acknowledgements

FlashCopy would not exist today were it not for the diligent work of a large development team led by Yoram Novick along with the support of the entire ESS development team.

References

[1] *A Dictionary of Storage Networking Terminology*, `http://www.snia.org/English/Resources/Dictionary_FS.html`

[2] Hutchinson, N., Manley, S., Federwisch, M., Harris, G., Hitz, D., Kleiman, S., O'Malley, S., "Logical vs. Physical File System Backup" *Third Symposium on Operating Systems Design and Implementation.*1999.

[3] Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., West, M., "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988, Pages 51-81.

[4] EMC TimeFinder Product Description Guide, 1998, EMC Corporation, `http://www.emc.com/products/product_pdfs/pdg/timefinder_pdg.pdf`

[5] Hitachi ShadowImage, June 2001, `http://www.hds.com/pdf/shadowimageR6.pdf`

[6] Mellish, B., Blazek, V., Beyer, A., and Wolatka, R. *Implementing ESS Copy Services on UNIX and Windows NT/2000*. Feb. 2001, IBM.

[7] McAuley, D., Pate, A., Black, I., Bueffel, V., Chana, B., Docherty, G., Leplaideur, D., Nel, W., *IBM RAMAC Virtual Array,* July 1997, IBM

[8] "StorageTek™ SnapShot Software"
http://www.storagetek.com/products/software/snapshot/

[9] Brown, K., Katcher, J., Walters, R., Watson, A., *SnapMirror and SnapRestore: Advances in Snapshot Technology,*
http://www.netapp.com/tech_library/3043.html, Network Apppliance, Inc.

[10] Gibson, G.A., R. Van Meter, "Network Attached Storage Architecture," *Communications of the ACM,* Vol. 43, No 11, Nov., 2000

[11] "Object Based Storage Devices: A Command Set Proposal,"
http://www.nsic.org/nasd/final.pdf Nov. 1999

[12] Menon, Jai. "A performance comparison of RAID-5 and log-structured arrays" Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing, 1995. p.167-178.

[13] Chervenak, A., Vellanki, V., Kurmas, Z., "Protecting File Systems: A Survey of Backup Techniques", Proceedings *Joint NASA and IEEE Mass Storage Conference,* March 1998.

[14] EMC TimeFinder, 2000, EMC Corporation,
http://www.emc.com/products/product_pdfs/ds/timefinder_l700-4.pdf

[15] Pate, A., Vaia, C., Todd, J., and Aigner, H. *Implementing DSMSdss SnapShot and Virtual Concurrent Copy,* June 1998, IBM

[16] Blunden, M., Bergum, S., Dovidauskas J., and Vaia, C. *Implementing ESS Copy Services on S/390,* Dec. 2000, IBM.

---

[i] Or tracks and volumes for zSeries; we focus on open systems.

Locating Logical Volumes in Large-Scale Networks

Mallik Mahalingam, Christos Karamanolis, Magnus Karlsson, Zhichen Xu

Hewlett Packard Labs

1501 Page Mill Rd, Palo Alto CA 94304

{mmallik, christos, karlsson, zhichen}@hpl.hp.com

Abstract

Storage is increasingly becoming a commodity shared in global scale, either within the infrastructure of large organizations or by outsourcing to Storage Service Providers. Storage resources are managed and shared in the form of *logical volumes*; that is, virtual disks that aggregate resources from multiple, distributed physical devices and storage area networks. Logical volumes are dynamically assigned to servers according to a global *resource utility model*.

This paper focuses on the problem of locating and accessing logical volumes in very large scale. Our goal is to devise mechanisms that are least intrusive to the existing Internet infrastructure. Two methods are proposed, based on *DNS name resolution* and *BGP routing*, respectively. The former is based on the current DNS protocols and infrastructure; the latter requires extensions to the existing BGP protocols. The two approaches are evaluated by means of simulations, based on realistic workloads and actual Internet topology. It is shown that the simpler and less intrusive DNS-based approach performs sufficiently well, for even small caches on the clients.

1 Introduction

Storage Service Providers (SSP) such as ScaleEight [1] and StorageNetworks [2] provide network-based storage solutions for customers that wish to outsource some or all of their data storage and its management. They provide a global storage infrastructure that enables their customers to create, manage and distribute large sets of data across multiple geographic locations.

Clients access such a global storage service in one of two ways. First, directly by means of traditional file system APIs, e.g., through NFS mount-points. These clients are typically hosts that execute application services for the organizations that outsource storage to the SSP. Second, by means of Content Delivery Networks (CDNs) [3, 4], which replicate certain types of the data (originating from the SSP) closer to the edge of the network. We envision that in future storage services, the borderline between SSPs and CDNs will be blurred, as content will be dynamically created and stored at the edge of the network. The emerging technologies for distributed application services [5, 6] and peer-to-peer CDNs [7] point in that direction. Throughout this paper, we use the term *clients* to refer to both these classes of clients.

Typically, the infrastructure of an SSP consists of a pool of storage resources, such as disks, disk arrays and Storage Area Networks (SANs), as well as compute resources (servers) for providing access to the storage. This infrastructure is physically distributed across multiple geographic locations. SSPs may own their own Data Centers, or their resources may be hosted at Internet Data Centers (IDCs), such as those of Exodus [8] and Qwest [9]. Moreover, we anticipate that, in the future, storage service providers

will not necessarily own their own physical resources. Instead, their infrastructure will be provided by on-demand aggregation of resources from multiple disparate data centers, following the principles of a *resource utility* model [10, 11].

Even today, the infrastructure of SSPs and big corporations consists of many, heterogeneous and distributed physical storage resources. In this context, *logical volume managers* are used in order to simplify the management and facilitate the use of diverse resources. *Logical volumes* provide an abstraction for aggregating storage resources spread across multiple disks (that are attached to the same server or the same SAN) to appear as a single virtual storage device [12]. Data is organized within the boundaries of the logical volumes. Data on volumes are accessed through one or more servers that mount that volume. The data may be organized in the form of a file system or a database. To keep the discussion simple, in the rest of the paper, we will refer to data as files.

Clients access a volume by going through the corresponding file server, which coordinates all accesses via a file system API. When a client requests access to a file (performs a lookup), a *file-handle*, which uniquely identifies the file in the system, is returned back to the client. This file-handle contains a *Volume Identifier (VID*) that refers to the logical volume where the file is physically stored [13, 14]. Files accessed by a client may be spread across multiple logical volumes. Therefore, for every file access, the client must resolve the location of a file server that "owns" the logical volume where the file resides.

In a resource utility model, the mapping of logical volumes to physical resources and their assignment to file servers can be dynamic. Therefore, a key problem is how to provide efficient and scalable mechanisms for locating a logical volume and its custodian file server. The system model we assume for our discussion is outlined in section 2. In section 3, we propose a mechanism by which file servers can locate the logical volumes that they are responsible for. Sections 4 and 5 introduce two mechanisms for resolving the identity of a server that provides access to a volume. The main idea behind the proposed solutions is to exploit well-understood mechanisms, with proven scalability in the Internet, and adapt them for locating volumes in very large scale. Our aim is to use existing services (e.g., DNS), with no or minimal changes to the existing infrastructure. The two approaches are evaluated in section 6, using simulation based on both real and synthetic workloads, as well as real Internet topology information. Section 7 discusses related work and section 8 concludes the paper.

2 System overview

The infrastructure of an SSP resembles any other network in the Internet. We assume it is divided into a number of *Zones*, each with a unique identifier, *Z-ID*. Each Zone consists of one or more Autonomous Systems (AS) and each Autonomous System consists of a number of Autonomous System Regions (ASR). *An* ASR representative maintains a database that contains information on the logical volumes within its region and their assigned servers. By organizing the system this way, we uniquely identify any logical volume by a Volume identifier (VID), using the convention "*Volume-ID.ASR-ID.AS-ID.Zone-ID*".

File servers typically retrieve their logical volume assignment by interacting with an ASR representative. The volume assignments may be dynamic to accommodate system reconfiguration, fluctuating demand or changing workloads. Automating the resource management in such environments is the focus of several current research projects [10, 11, 15].

When a client requires access to a file, it performs a lookup by sending a lookup request to the file server that hosts the logical volume where the parent directory of the file resides. The file server performs lookup locally on the parent directory and returns the file handle corresponding to the file. Note, that the volume (and server) of the parent directory, where the file handle is constructed, and the volume of the file itself may not be the same, as it is the case in systems such as Archipelago [16] and DiFFS [14]. The file-handle contains a *Volume Identifier (VID*) that refers to the logical volume where the file is physically stored. In order for the clients to access the file, they must resolve the VID and locate the file server that "owns" the corresponding logical volume.

3 Assignment of logical volumes to servers

When a file server comes online, it sends out a request identifying itself, asking for logical volumes that it is responsible for. This functionality is implemented using the DHCP protocol [17]. When an ASR representative within the vicinity of the file server receives the request, it locates the list of logical volumes that the requestor is responsible for and responds back supplying the list to the server. The response contains the configuration information of the logical volumes. For example, in an IP-based SAN, the response may contain Logical Unit Numbers (LUN) and their corresponding target IP addresses, along with other information such as whether a logical volume is stripped, mirrored, etc. The assignment of logical volumes may be pre-configured via storage management tools or may be dynamically assigned by an ASR representative upon receiving the request. Once an assignment is made, the representative for the ASR updates its database to reflect the new state of server-to-volume assignment. These assignments can be dynamically changed to cater for various system conditions such as file server utilization, load balancing, locality, etc. Any reassignment of logical volumes affects only the database of a specific ASR and leaves the rest of the mapping in the system intact.

In very large systems following the resource utility model, we cannot assume that file servers can reach ASR representative via DHCP. Two solutions can be applied in such environments: 1) the file server is pre-configured with a set of logical volumes; 2) the file server is configured with the identity of an ASR representative (not necessarily of its local ASR) which it should contact to retrieve its volume assignments.

4 Logical volume discovery by clients using DNS

In this approach, each *Zone, AS and ASR* has one or more designated representatives, which, in practice, are part of the existing DNS infrastructure (authoritative servers) [18]. The root server of the SSP contains information on all zone representatives. Every zone representative maintains a database with all the AS representatives within its zone.

In the same way, an AS representative maintains information about all ASR representatives within that AS.

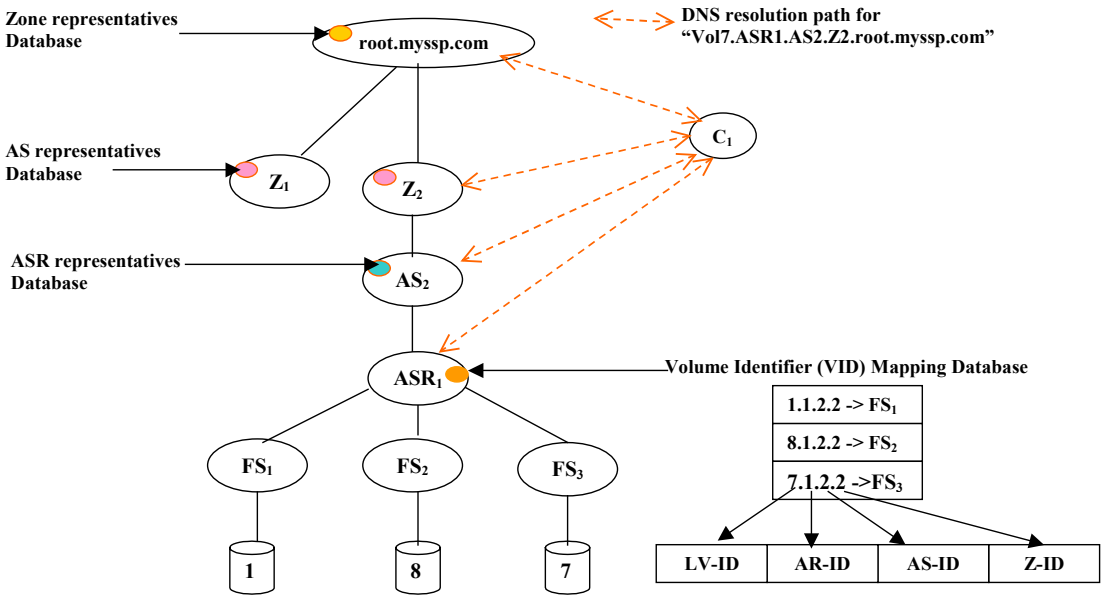


Figure 1: VID resolution using DNS

For a client to access a file, it has to first retrieve a file handle via a lookup process. The client then needs to locate the file server that corresponds to the Volume Identifier (VID) in the file handle. The identity of the server is resolved by exploiting typical DNS name resolution [18]. For example, when a client C_1 receives a file handle that contains VID *7.1.2.2*, it constructs a fully qualified domain name "Vol*7*.ASR*1*.AS*2*.Z*2*.root.myssp.com*" based on the numerical VID contained in the file handle and the root domain name of the SSP. The root domain name is obtained during the file system mount time The client then resolves this (artificial) domain name through a normal DNS resolution process, as depicted in Figure 1. This process does not require any changes to the existing DNS infrastructure. However, the root server of the SSP needs to be configured to respond to the domain suffix "Z2" by specifying the authoritative representatives for that part of the domain suffix. When a client's requests land at the representative for an ASR, the address of the file server that corresponds to the VID is returned. Results of this query can be cached at the client for improved performance.

Various optimizations are possible in order to speed up the resolution process. One possibility is to have file servers resolve the logical volume mapping, cache the information locally and return the mapping information when a file handle needs to be returned back to the client. This cached information could significantly reduce the network traffic especially when many clients reference the same logical volume. Cached information can be kept loosely consistent with the actual mapping by performing periodic checks. Also, resolution at the file server can be performed in an asynchronous fashion to hide any extra delays. Invalid references can arise due to volume reassign-

ments or the non-availability of file servers. In this case, clients resort back to the normal resolution process.

Clients can also contact a local DNS server and have that server perform the logical volume to file server mapping. Typically, employing optimizations like this has proven to produce higher cache hit ratio [19] in resolving domain names at the client.

5 Logical volume discovery using suffix-based routing

This section introduces an alternative approach for clients to retrieve the custodian file server of logical volumes, called *Volume Identifier Routing Protocol* (VIRP). Given a VID, VIRP routes the request for VID resolution to the corresponding ASR representative taking the shortest ASR (or AS) path and returns the address of the corresponding file server to the client.

VIRP is based on suffix reachability that is similar to prefix-based routing performed using BGP [20]. There are two variations of the protocol. In the first variation, each ASR representative advertises itself to its neighboring VIRP routers. These advertisements are propagated further to other VIRP routers. For a particular VIRP router, routing advertisement of an ASR representative indicates the shortest path towards that ASR representative.

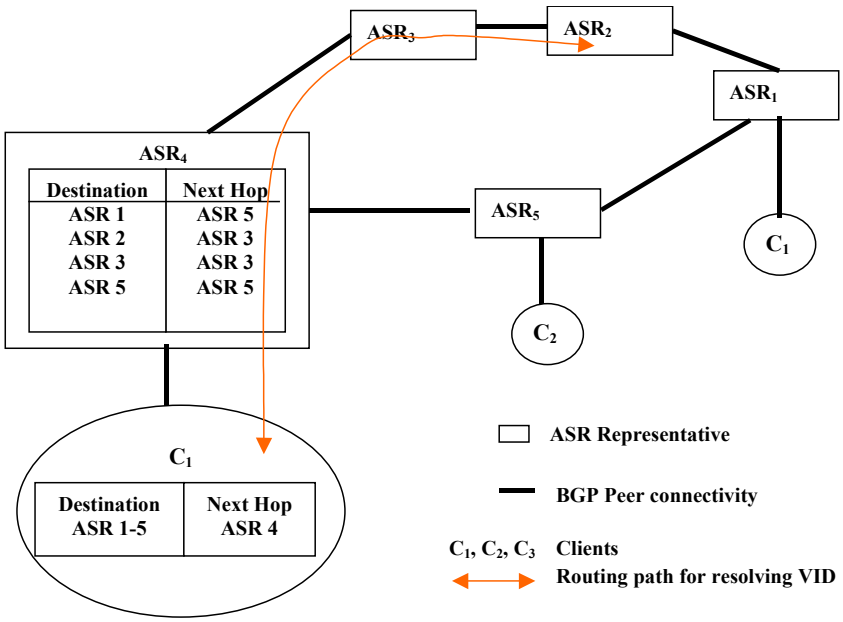


Figure 2: Example showing VIRP advertisements and routing VID resolution

For example, Figure 2 shows the routing table at VIRP router ASR_4. The routing table contains the next hop address for other ASR representatives following the shortest path. As shown earlier in section 2, VID contains a Volume ID, an ASR ID, an AS ID, and a Zone ID. Clients resolve VID by routing the request to the ASR representative corresponding to the ASR part of the VID. The routed request takes the shortest path leading to the target region. For example, a client C_1 that wishes to resolve a VID that belongs

to ASR_2 will first route to ASR_4 and then take ASR_3 as the following hop and route to ASR_2. In VIRP, the clients receive routing advertisements but do not perform any advertisements.

Alternatively, to reduce the size of VIRP routing tables, the advertisement can be performed at the AS level. We introduce a representative for each AS to receive requests from clients and direct them to ASR representatives. The AS representatives advertise themselves as it was done in the previous case. Once a client request is routed to an AS representative, the latter can forward the request to an ASR representative by performing a local lookup using the ASR-ID. The respective ASR representative responds to the client with the address of the file server using the volume part of the VID. This greatly reduces the number of entries kept in the routing tables but it requires defining additional protocols for interaction between AS and ASR representatives. To give the readers an idea of the savings on routing table size, assume that an ASR corresponds to a network prefix on the Internet. There are 150K unique prefixes whereas the number of AS on the Internet is on the order of 10K.

There are several ways to deploy this type of infrastructure. One way is to reuse the existing BGP routing infrastructure by adding new protocols. A more practical way is to construct an *overlay network* to build this infrastructure [21]. Such an overlay network can be constructed at application level for easy deployment.

6 Evaluation

The performance of the proposed DNS-based and BGP-based approaches is evaluated by means of simulations. The simulation model is based on an Autonomous System (AS) view of the actual Internet topology as of October 2001, and a real-world, globally distributed workload. We chose this to be a web workload for two reasons. First, we believe that Content Delivery Networks will be one of the main applications of a globally distributed file system, and secondly, it is one of the few workloads that today have millions of globally distributed clients. The metric used to compare the two approaches is *client perceived latency* in resolving a VID.

6.1 Simulation Methodology

Our simulation model uses three sets of inputs in order to calculate the client perceived latency for the approaches: An Internet topology, a set of volumes and their locations, and finally the location of the clients and a list of chronologically ordered accesses to these volumes. The input parameters are all summarized in Table 1.

The Internet topology was generated using BGP routing table information obtained from a leading ISP, *Telestra.net* [22], during October 2001. From these routing tables an undirected graph is constructed, in which nodes represent Autonomous Systems and edges represent their peering relationship. The generated graph contains approximately 13.000 nodes and 150.000 edges and we assume a uniform edge cost. The distance between two nodes in the topology is measured in number of AS-level network hops on the shortest path between those nodes. The placement of the DNS servers in this Internet topology is decided in the following way. We generated a list of nodes sorted in descending order of their fan-out (number of nodes that are just one hop away from one

specific node). The node that has the highest fan-out is selected to be the representative for "*root*" and removed from the list. Next, the set of zone representatives are picked from the top of the list and then are removed from the list. The AS and ASR representatives are chosen in the same way.

Table 1: The main parameters of the experimental platform and their corresponding values. The shaded parameters are the ones that we vary in the experiments.

	Parameter	Value
Topology	Distribution	Part of real Internet
Volumes	Number	20,000 or 80,000
	DNS nodes	4/10/5/100 (Z/AS/ASR/Volumes) or 4/40/5/100
Objects	Number	90,000 or 1 million
	Distribution	Sequential or Random
Clients	Number	5,400 Client clusters
	Distribution	According to real AS location
	VID access pattern	WorldCup98 or Random

The object references were obtained from web logs of the World Cup Soccer 1998 event [23]. The logs contain references to nearly 90K unique files. These files are mapped on 20K and 80K volumes, respectively for the two scenarios. While clearly the World Cup site would not in reality be located on this many volumes, a client would not access solely one site. Instead a client would be accessing many different volumes of various sites. Our client workload can thus be seen to represent a widely scattered surfing pattern that is close to a worst-case scenario for the DNS approach. The placement of objects to volumes is done in two ways: *sequential* and *random*. For each of these algorithms, N files (where N = unique files / no of nodes) need to be placed on each volume. For the sequential algorithm, the first N unique files encountered in the web log are placed on node 1.1.1.1. The following N unique files are then placed on node 2.1.1.1, and so on. As more frequently accessed files tend to show up earlier in the web log, this algorithm will place popular files closer to each other. The random algorithm, on the other hand, places the first N files encountered in the web log on a random node, the next N files on another random node, and so on.

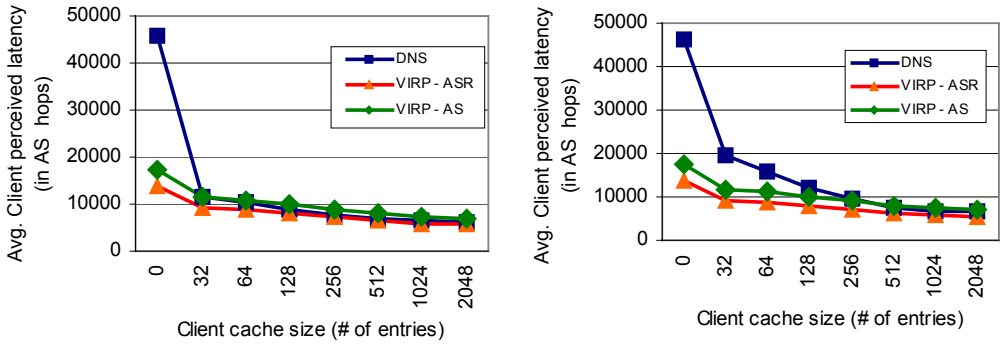
The clients' locations and access patterns were also obtained from the 98 World Cup logs. These contain accesses made by roughly 2.6 million clients over the course of 90 days (includes accesses made 30 days prior and 30 days after the event). To be able to assign these clients to the AS node they actually resides on in reality, we developed a program that converts IP address of a client to the corresponding AS ID. This clustering generated about 5.4K unique client clusters that are located in the same number of unique ASs.

We use two different client access traces to evaluate the proposed schemes: *World-Cup98* and *random*. The former is taken straight from the client accesses of the World Cup log; the latter is a uniformly random VID accesses. In the World Cup log, all clients in one AS access, on the average, 1K unique objects, while in the random one, the simulation is terminated after 2K unique objects are referenced by each AS.

To measure the client perceived latency, 20% of the ASs were randomly chosen and used in the simulations. They represent 500K clients generating close to 20% of the total client accesses. For each AS, a list of objects that the clients in that AS accessed is generated. In our model, every server (DNS server or VIRP router) that is queried adds to the client perceived latency. We express the client perceived latency in terms of the number of AS hops involved. This has been shown to be a fair measure of latency [24]. Network contention is not taken under consideration. For the simulation, we have used simple LRU caching at the clients to store the resolved VIDs. The impact of the size of this cache and all other shaded parameters in Table 1 are examined in the next section.

6.2 Performance Results

The initial intuition was that the DNS approach should have a higher client perceived latency than the VIRP approaches, when the VID lookup cache size is small and/or when the locality of VID lookups is poor. In this section, we will investigate how much locality the DNS approach needs in order to be comparable to the VIRP approaches, and provide a rough estimate on how many VID lookups need to be cached at each client for this to be achieved.

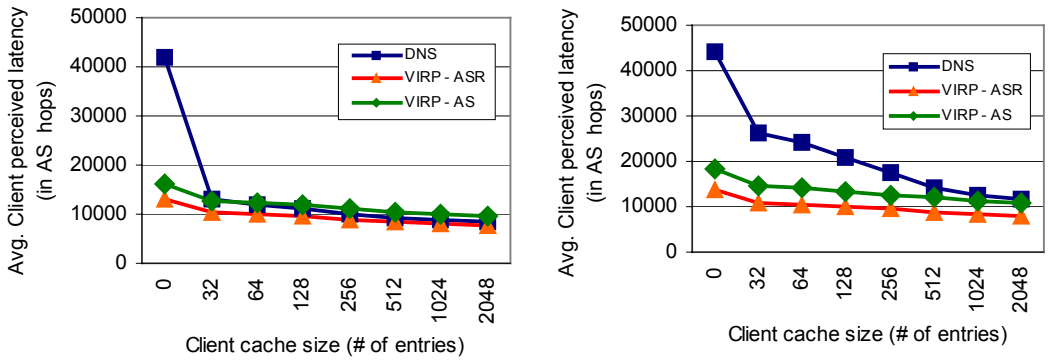


(a) Sequential object distribution ***(b) Random object distribution***

Figure 3: Results for the DNS, VIRP-ASR and VIRP-AS approaches. Number of volumes: 20,000. Number of objects: 90,000. Client access pattern: WorldCup98.

Figure 3a shows the results for the DNS, VIRP-ASR and VIRP-AS approaches using sequential object distribution. In the figure, the x-axis represents the various client cache sizes and the y-axis represents the average client perceived latency due to the VID lookup process. VIRP-ASR has the lowest client perceived latency as it requires only one lookup message and it traverses the shortest path between the client and the server. For VIRP-AS, there is a potential for one more message, thus the slightly worse

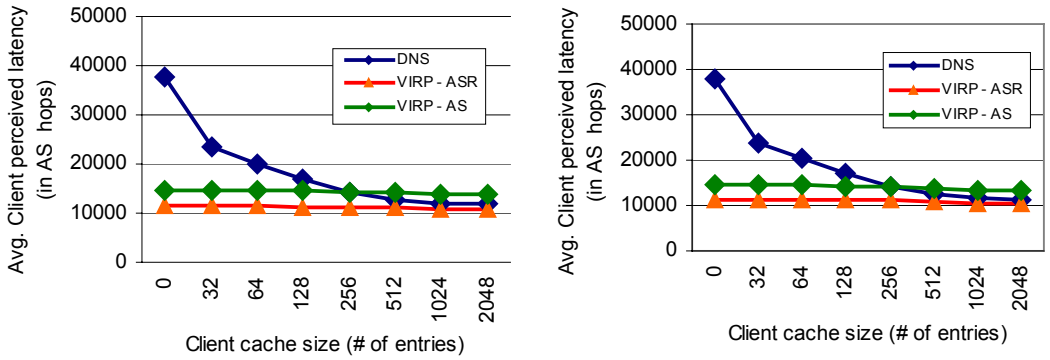
performance. The most interesting point in this graph is that the DNS approach performs well even for small client cache sizes. For the sequential object distribution of Figure 3a it starts to perform well at 32 entries, but for the random case in Figure 3b, this point is only increased to 256 entries. For a straightforward implementation of the client cache, this translates to a modest 1KB and 8KB of memory space, respectively.



(a) Sequential object distribution *(b) Random object distribution*

Figure 4: Results for DNS, VIRP-ASR and VIRP-AS approaches. Number of volumes: 80,000. Number of objects: 90,000. Client access pattern: WorldCup98.

Figure 4 shows the effects of what happens if the number of volumes is increased four times to 80,000 volumes. As the locality will be poorer than before, we would expect the DNS approach to perform even worse. But for the sequential object distribution it hardly matters for clients with a cache, as the DNS approach performs as well as before. However, for the random object distribution the cache size required for DNS to become comparable to VIRP-AS is larger. It is now around 2K entries, translating into 64KB of memory space.

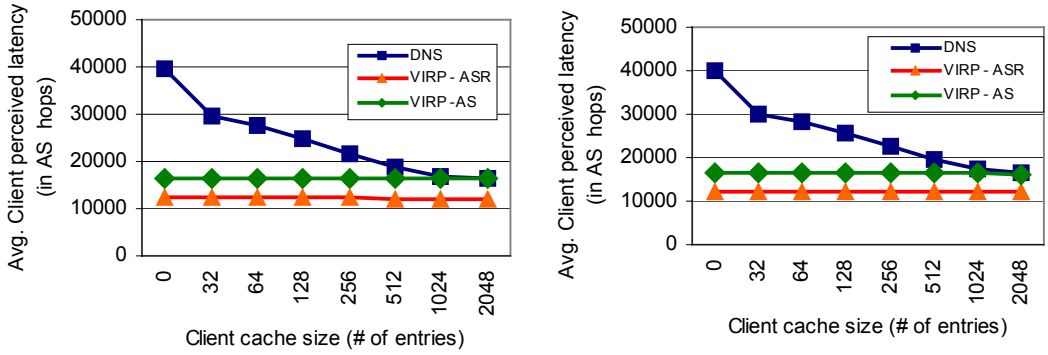


(a) Sequential object distribution *(b) Random object distribution*

Figure 5: Results for DNS, VIRP-ASR and VIRP-AS approaches. Number of volumes: 20.000. Number of objects: 1 million. Client access pattern: Random.

The last set of experiments was designed to stress the approaches even further to see how they hold up for a random client access pattern with a larger number of objects.

Few workloads will have access patterns that are truly random, however, this will provide us with a worst-case scenario for the approaches. Figure 5 and 6 show the results for the random client-access pattern when the number of objects is 1 million. It can be seen that the VIRP approaches perform better than the DNS approach for small sizes of caches, but their performance remains more or less unaffected by the client cache size.



(a) Sequential object distribution *(b) Random object distribution*

Figure 6: Results for DNS, VIRP-ASR and VIRP-AS approaches. Number of volumes: 80.000. Number of objects: 1 million. Client access pattern: Random

This is due to the random accesses to volumes. There is little reuse of VIDs as the lookups are completely random, thus there is also little use of the client cache for storing individual VID lookups. However, for the DNS approach there will still be access locality for the entries that store the zone, AS and ASR lookups as there are far lower number of these in the system than volumes. This explains why DNS benefits from a larger cache but not the VIRP approaches for this experiment. Thus, even for modest cache sizes, the performance of the DNS approach is comparable to that of VIRP.

6.3 Summary of simulation results

Our simulation shows that VIRP with ASR level aggregation outperforms all other approaches we compared against. The drawback with the VIRP approaches is that they require protocol modifications to the existing routing infrastructure. The DNS approach, on the other hand, can be deployed on existing infrastructure. Its performance is comparable to VIRP for reasonable client cache sizes even when the locality is poor. For reasonable cache sizes, the type of the object distribution has lesser effect on the client perceived latency. In general, we believe that the deployment of the DNS approach is preferable as its performance is comparable to the VIRP approaches, while using existing infrastructure.

7 Related Work

Existing distributed storage systems, such as AFS [13, 14], are designed for deployment in campus environments. These systems maintain a *volume location database* (VLDB) to track the servers in the system where volumes reside. For example, AFS maintains a VLDB for every "cell" of the system. The VLDB is typically replicated on

two or more *Volume Location Servers*, for availability reasons. An AFS client within a cell is manually configured with a list of *Volume Location Servers* that it can contact to resolve the volume location. This is not a feasible choice for large-scale geographically dispersed networks such as the Internet. Also, AFS does not provide any mechanisms by which *file servers* can locate the logical volumes they are assigned to; this information is hard-wired in the servers' configuration.

Volume managers such as that of Veritas [25],[26] and storage virtualization systems [27] aggregate multiple, disparate physical storage resources using the volume abstraction. These solutions are applicable to small-scale systems, a single SAN and a single data center. Neither they provide service for hosts in the network to discover their assignments nor they allow clients to resolve the owners of logical volumes.

Techniques used by SSPs such as Scale8 [1] are not published. Karamanolis *et al*. [14] describe mechanisms by which a file server keeps limited information about the peers that the logical volumes under its custody have references to. Their proposal is primarily an optimization of our DNS approach, where caching is used at the file server.

8 Conclusion

Storage is increasingly becoming a commodity resource shared in global scale. The emerging business model of outsourcing storage (or its management) to third-party service providers amplifies this trend. In this context, storage resources are virtualized and shared by means of logical volumes. This paper addresses the problem of locating and accessing logical volumes in global infrastructures, as those of Storage Service Providers or large corporations.

The paper briefly describes ways to assign computational resources (servers) to volumes and how this mapping is performed in various system models. We then focus on mechanisms for clients to locate and access logical volumes, in a very large, dynamic infrastructure. That is, locate the servers that provide access to specific volumes. In environments of the scale and volatility required in a "resource economy", a centralized volume location database does not provide a satisfactory solution. First, it does not scale sufficiently (e.g., for tens of thousands of volumes); second, we cannot expect a centralized "knowledge" of the entire system's configuration.

The motivation for the work presented in this paper was to investigate solutions that are based on well-understood and provably scalable mechanisms. In that spirit, two approaches are proposed to address the problem. The first is based on existing DNS infrastructure and protocols to resolve hierarchical volume identifiers. The second proposes extensions to existing BGP routing protocols to efficiently locate host servers of volumes.

Our initial assertion was that the BGP-based approach would perform better than the DNS approach. However, experimental results based on simulations indicate that even for modest volume-id caching on the clients, the benefits of BGP are negligible. Moreover, the DNS approach is based completely on existing protocols and it is not intrusive to the existing infrastructure. So, its deployment would be straightforward. On the other hand, the BGP approach requires extensions to existing protocols and routing table

management, making it much harder to be deployed in a real environment. The latter is not justified by the marginal performance benefits this approach offers.

9 References

[1] ScaleEight, http://www.scale8.com/.

[2] StorageNetworks, http://www.storagenetworks.com/.

[3] Akamai, http://www.akamai.com.

[4] DigitalIsland, http://www.digitalisland.com.

[5] Ejasent, http://www.ejasent.com.

[6] Zembu, http://www.zembu.com.

[7] J. Kangasharju, J. W. Roberts, and K. W. Ross, "Object Replication Strategies in Content Distribution Networks," presented at 6th Web Caching and Content Distribution Workshop, Boston, MA, USA, 2001.

[8] Exodus, http://www.exodus.com.

[9] Qwest, http://www.qwest.com.

[10] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. P. Pazel, J. Pershing, and B. Rochwerger., "Oceano - SLA Based Management of a Computing Utility," presented at Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management, 2001.

[11] J. Wilkes, J. Janakiraman, P. Goldsack, L. Russell, S. Singhal, and A. Thomas, "Eos - The Dawn Of The Resource Economy," presented at HotOS-VIII Workshop, Schloss Elmau, Germany, 2001.

[12] D. Teigland and H. Mauelshagen, "Volume Managers in Linux," presented at FREENIX Track: 2001 USENIX Annual Technical Conference, Boston, Massachusetts, USA, 2001.

[13] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West, "Scale and Performance in a Distributed File System," *ACM Transactions on Computer Systems*, vol. 6, pp. 51-81, 1988.

[14] C. Karamanolis, L. Liu, M. Mahalingam, D. Muntz, and Z. Zhang, "An Architecture for Scalable and Manageable File Services," Hewlett-Packard Labs, Palo Alto, Technical Report HPL-2001-173, July 2001.

[15] IBM, "Autonomic computing," : http://www.research.ibm.com/autonomic.

[16] M. Ji, E. W. Felten, R. Wang, and J. P. Singh, "Archipelago: An Island-Based File System for Highly Available and Scalable Internet Services," presented at 4th USENIX Windows Systems Symposium, 2000.

[17] DHCP, http://www.dhcp.org.

[18] DNS, http://www.dns.net/dnsrd/.

[19] E. Sit, "Study of caching in the Internet Domain Name System," Massachusetts Institute of Technology, May 2000., 2000.

[20] Y. Rekhter, T. Li, and M. 1995, "A Border Gateway Protocol 4 (BGP-4) - RFC 1771," in *Request for Comments: 1771*, 1995.

[21] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz., "Bayeux: An Architecture for Scalable and Fault-tolerant WideArea Data Dissemination," presented at In Proc. of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001), 2001.

[22] Telstra, "Raw BGP Data," http://kahuna.telstra.net/bgp2.

[23] Worldcup98, "Worldcup98 soccer event - Web logs," : http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[24] P. Radoslavov, R. Govindan, and D. Estrin, "Topology-Informed Internet Replica Placement," presented at 6th Web Caching and Content Distribution Workshop, Boston, MA, USA, 2001.

[25] VERITAS, "Veritas Volume Manager," http://www.veritas.com.

[26] M. Hasenstein, "The Logical Volume Manager (LVM)," http://www.sistina.com/lvm_whitepaper.pdf.

[27] StorageApps, "SANLink," http://www.hp.com/products1/storage/san/sanlink/index.html.

Building a Massive, Distributed Storage Infrastructure at Indiana University

Anurag Shankar, Gerry Bernbom
University Information Technology Services
Indiana University
2711 East Tenth Street
Bloomington IN 47408
ashankar, bernbom@Indiana.Edu
tel: +1-812-855-9255
fax: +1-812-855-8299

Abstract
Anticipating an onslaught of data in research, administrative, and academic computing, Indiana University (IU) undertook in 1998 the ambitious task of architecting a massive, distributed storage infrastructure to meet its long-term storage needs. The task, now nearly complete, has resulted in the institution of the High Performance Storage System (HPSS), a hierarchical storage management (HSM) system, augmented by the Distributed Computing Environment Distributed File System (DCE DFS) acting both as a file system front end to HPSS and as a common file system (CFS) for IU campuses. Using gateways, IU's distributed storage system today currently offers a user on its eight geographically distributed campuses a capacity for securely storing and accessing nearly 200 Terabytes of data from any networked (Windows, Mac, or Unix/Linux) desktop equipped with a web browser.

HSM systems such as HPSS have traditionally been used by high-end users at large research labs (for example Los Alamos, Livermore, Sandia, Brookhaven National Labs in the U.S. and at CERN in Europe), at supercomputer centers (for example the San Diego Supercomputer Center), and at government agencies such as NASA. IU's installation is unique in two respects. It is the first production HPSS that is geographically distributed over a wide area network (WAN). Second, we have made available a high-end storage system in an academic setting not only to traditional, high-performance research users (for example astronomers, physicists, chemists, etc.), but also more generally (to users in economics, fine arts, apparel design, music, libraries, life sciences, etc.).

1 Re-centralization of Storage

Why build a centralized data storage system when typical personal computer hard disks today offer tens of gigabytes of storage at a very low (acquisition) cost? While it is certainly true that the availability of cheap, abundant personal storage capacity in the early nineties started a trend toward de-centralization of storage (from a highly centralized mainframe era), this trend is slowing. Researchers on university campuses a decade ago found to their delight that, for the most part, they were able to acquire (through grants) the resources necessary to store their data locally, on personal workstations or on servers in their offices or in departments. However, their initial enthusiasm soon dissipated when the high, after-purchase cost and effort of ensuring the integrity, protection, and long-term storage of data became apparent. As hard disk drive sizes have swelled to gigabytes and then to tens of gigabytes, backups have become increasingly costly, even painful. Also, enterprise-wide, the need for protecting

institutional intellectual assets (in the form of research and other data created by users) has grown progressively stronger over the past decade, forcing many institutions to reconsider centralizing data storage.

2 Infrastructure Choices

The design of a storage infrastructure ultimately depends on a number of factors, chief among which are a) the amount of data to be stored, b) user data access patterns, and c) the available budget. With disk prices continuing their free fall, the storage industry seems to have agreed on storage area networks (SANs) to provide redundantly configured disk-based storage. However, SANs or alternatives that utilize spinning disks alone are simply not cost effective in building petabyte class data repositories at the present time. This leaves us with tapes and with HSM technology. The largest data repositories in the world are thus built using HSM systems. The tape to disk price ratio per megabyte of storage (especially at the high end) still favors tape over disk.

In a traditional HSM system, data bits are migrated seamlessly (from a user's perspective) from finely tuned, fast but (relatively) small disk caches (ours is a TB) to massive tape libraries (again, ours offers 200TB) when unused for a period of time. Metadata resides on disk forever (and is backed up carefully and redundantly). The user pays a price for having easy access to terabytes of data in the form of tens of seconds to possibly minutes-long delay in retrieving data bits that have migrated to tape. However, this appears to be acceptable for the majority of academic users who are happy to have access to massive storage capacities normally outside the scope of their individual or departmental budgets.

Armed with this information, we began looking for a HSM solution that provided a) long-term vendor viability, b) excellent hardware and software support, c) scalable performance, d) ease of access (preferably via a file system), and e) the ability to distribute software and hardware components geographically. At the conclusion of our request for proposal (RFP) process, only one contender remained, namely the High Performance Storage System. The HPSS[1] is the result of a fruitful collaboration between a number of government labs, academia, and IBM. It is not a vended solution in the usual sense; one pays instead a membership fee to join the HPSS collaboration. Each member gains access to the source code and is free to modify it within the mechanisms provided by the collaboration. Excellent software support is also included. Another attractive feature of HPSS is its ability to present a file system interface to data stored on tapes via DCE DFS[2], a distributed, scalable and secure file system.

At the high end, campus projects needing massive data storage at IU included candidates such as next generation high-energy physics experiments, with the potential to generate petabytes of data each year. With possible analysis times extending to decades, protection against software and hardware obsolescence is paramount. We felt that HPSS fit these needs and our environment well, by giving us long-term control over our destiny. [HPSS is also the HSM system of choice at some of the world's largest data repositories (for example SDSC, Brookhaven National Labs, CERN, etc.).]

Finally, while a tape-based system is ideal for archiving large files (tapes perform best when streaming), many campus users needed persistent, disk-based storage as well. In the past, this need was met (though inadequately) by the Novell Netware file system. By 1999 however, the future of Novell itself was in question and the existing Novell infrastructure was in urgent need of repair or replacement. With DCE DFS software already installed for HPSS purposes, it was natural to use it in lieu of Novell. DCE DFS is one of the most highly scalable and distributed file systems in use currently in the industry, to deliver high-end, secure file service. However, since DCE DFS clients are available only for a number of Unix flavors and for Windows NT4, it was clear that appropriate gateway servers would be needed to extend DCE DFS to the pervasive base of Windows and Mac desktops and servers on campus.

3 Building IU's Distributed Storage System

Our service design included campus users (using their personal workstations or departmental servers or our supercomputers) who either required massive, archival storage and/or who needed traditional, disk-based storage. A major design goal for us was also to provide storage ubiquitously, either via the web or via a file system front-end. Though these methods do not provide the highest performance, they were targeted for a non-savvy computer user due to the simplicity of use.

The majority of our users were located on two of IU's eight campuses, namely IU Bloomington (IUB) and IU-Purdue University at Indianapolis (IUPUI), located around fifty miles apart in central/south-central Indiana. Since the intercampus bandwidth (45Mbps) was insufficient to move massive amounts of data between campuses, we decided to experiment distributing IU's HPSS hardware and software across the two campuses. While the metadata engine remained at IUB, the actual user data was to reside where the user was located physically, either at IUB or at IUPUI. The idea was to use the intercampus link efficiently, to carry metadata traffic only. Users were to access their data over their local LAN at each campus via third party transfers. Extensive tests in partnership with IBM validated the idea and the experiment transformed into the first production instance of a remote HPSS mover at IUPUI in late 2000.

The file system front-end to HPSS is configured via "migrating" DFS servers. Data placed into HPSS via DFS arrives first in the DFS server disk caches, and later migrates to HPSS disk caches via a bi-directional DFS-HPSS link. The migrating DFS is thus a dedicated, external subsystem to HPSS. Static (i.e. non-migrating) DFS was also configured using separate DFS servers, with no link to HPSS, to provide the "Common File System" (CFS) service to the masses (directly, via DFS clients, and via SMB, Appleshare IP, and web gateways). Security for both HPSS and for CFS is provided through DCE (based on Kerberos 5).

We configured our core HPSS on a dedicated IBM RS/6000 SP located at IUB. This allows the eleven PowerPC "Silver" thin and wide SP nodes (which run core HPSS servers, disk/tape movers and migrating DFS servers) to communicate over the IBM SP switch at 130MB/s. Our supercomputer (another IBM SP) users are able to transfer data to/from HPSS using an ASCEND router at better than 100MB/s. A terabyte of IBM's

serial storage array (SSA) disk attached to the eleven nodes forms the HPSS and migrating DFS disk caches. We use IBM's Magstar (3590E) tape drives in an IBM 3494 tape library and Storage Technology Corporation's 9840 "Eagle" tape drives in a STK 9310 tape library to store HPSS data at IUB. Remote HPSS disk and tape movers and a DFS server are configured on an IBM H70 server at IUPUI in Indianapolis. We have roughly 1TB of UltraSCSI RAID5 disk configured on the H70 as HPSS and migrating DFS disk caches. A number of IBM's Magstar drives inside an IBM 3494 tape library are SCSI-attached to the H70 at IUPUI.

HPSS is accessed in a high-performance mode via especially written Unix clients or more easily via FTP, DFS or via the web. We currently have around a thousand users distributed across various IU campuses, with roughly 55TB of data stored in HPSS.

IU's non-migrating DFS (or CFS) runs at IUB on several IBM's low-end B50 servers with IBM's UltraSCSI RAID5 arrays. Five Sun E220R servers run Samba[3], Netatalk[4], and Apache-SSL[5] servers which allow Win9x, Mac, and Linux users to access DFS from any networked desktop. The gateways are accessed by users as a single, round-robin DNS name. User authentication occurs securely (via modifications to Samba, Netatalk, and Apache server code[6]) directly against DCE. This allows no name space information to be maintained on the gateways, thus helping load balance and scale the service up as appropriate, without user impact. The non-migrating DFS servers and the gateways together form our CFS environment which is available to all campus users, either as a mapped drive under Windows, an an Appleshare IP volume on Macs, via smbmount or a native DFS client under Unix (or smbfs under Linux), and via the web. We are serving roughly 25,000 CFS customers currently with 250GB of data stored and backed up regularly.

4 Future

We are currently working in partnership with IBM to investigate developing an interface between IBM's high-performance general parallel file system (GPFS) and HPSS. This could enable high-speed, parallel, file system based data transfers between Linux clusters and HPSS (these clusters are currently served largely via low-performance NFS). We are also expanding the HPSS infrastructure at our Indianapolis campus (to nearly 400TB capacity) to support life sciences research and will start tests soon thereafter to mirror all HPSS data in real time across i-light[7], a newly installed high-speed optical fiber infrastructure between IUB and IUPUI. This should provide us with better protection against a disaster at either site. Finally, CFS is being extended to the IUPUI campus and will replace the local Novell infrastructure during 2002.

5 Conclusions

Indiana University is one of the few academic institutions to successfully anticipate and to build an ambitious infrastructure to provide massive data storage to its users. Using HPSS, a highly scalable and distributed hierarchical storage management system, along with DCE DFS and SMB, AppleShare IP and web gateways, a campus user at IU can store and access terabytes of data from their desktops. We have also found that it *is*

possible to implement and to offer a high-end storage system to the masses, with significant cost savings over the long run.

We are happy to share our knowledge and experiences with anyone who is interested.[8]

References

[1] Information about the High Performance Storage System (HPSS) is available at the website http://www.clearlake.ibm.com/hpss/.

[2] IBM's DCE website: http://www.ibm.com/software/network/dce/. IBM's Transarc Labs DCE/DFS website: http://www.transarc.ibm.com/Product/.

[3] Samba project website: http://www.samba.org/.

[4] Netatalk project website: http://www.umich.edu/~rsug/netatalk/.

[5] Apache project website: http://www.apache.org/.

[6] Paul Henson's mods for Samba/Netatalk/Apache are available at the URL http://www.intranet.csupomona.edu/~henson/www/projects/.

[7] Indiana's high-speed research network website: http://www.i-light.iupui.edu/.

[8] Information about IU's distributed storage services is available at the URL http://storage.iu.edu/. Our distributed storage services group website address is http://www.indiana.edu/~dssg/.

High-density holographic data storage with random encoded reference beam

Vladimir B. Markov
MetroLaser, Inc.
18010 Skypark Circle, Suite 100
Irvine CA 92614
vmarkov@metrolaserinc.com
tel: +1-949-553-0688
fax: +1-949-553-0495

Abstract
Holographic technique offers high-density data storage with parallel access and high throughput. Several methods exist for data multiplexing based on the fundamental principle of volume hologram Bragg selectivity. We recently demonstrated that spatial shift selectivity associated with a random (amplitude-phase) encoding of the reference beam is an alternative method for high-density, high capacity data multiplexing. In this report we show some characteristics of the random encoded reference beam hologram selectivity[1].

1 Introduction

Volume holographic memory allows for high throughput data storage and retrieval. Different techniques for data multiplexing have been explored, including those based on angular [2] and spectral [3] selectivity of volume holography, spatial encoding of the reference beam [4] or a combination of these methods [5]. The combination of reference beam phase encoding with spatial-shift multiplexing was shown to be an efficient approach for high-density holographic information storage [6,7]. The correlation effects at volume hologram recording and reconstruction with random encoded (speckled) reference beam came out as the part of the analysis of the holographic laser beam corrector [8]. A similar technique using a reference beam comprised of many plane waves (or a spherical wave) was suggested and experimentally demonstrated [9]. In this report some characteristics of volume hologram with random-encoded reference (**RER)** beam are discussed.

2 Theoretical Analysis

In our analysis we consider a volume hologram recorded by a plane wave signal beam $\mathbf{S_o(r)}$ and a RER-beam, $\mathbf{R_o(r)}$, with a divergence $\delta\theta_{SP}$. By intersecting at an angle θ_o these two beams form a hologram with average grating spacing $\Lambda = \lambda/sin(\theta_o)$, assuming an incidence angle $\theta_{Ro} = 0$. In the first Born approximation, the diffracted beam amplitude $\mathbf{S(r)}$, when reconstructed with RER-beam different from the recording one i.e. $\mathbf{R(r)} \neq \mathbf{R_o(r)}$, can be described as [10]:

$$S(r) = k_o^2 \iiint_V \delta\varepsilon(r')R(r')\frac{\exp[ik_o(r-r')]}{4\pi|r-r'|}d^3r' . \qquad (1)$$

Here $\delta\varepsilon(r) \propto S_O(r)R_O^*(r)$ is the recording media permittivity modulation and **V** is the volume of the hologram with thickness **T**. Eq. (1) is valid if $\mathbf{T} >> \lambda/(\delta\theta_{SP})^2$, i.e. exceeds the longitudinal speckle size.

We introduce now the normalized diffracted beam intensity $\mathbf{I_{DN}(\Delta)}$ as the parameter to describe the selectivity properties of RER-beam hologram:

$$I_{DN}(\Delta_\perp) = \frac{I_D(\Delta_\perp)}{I_{D(\Delta_\perp=0)}} = \left| \int_0^T \exp\left[\frac{ik_o n \Delta_\perp^2}{2d_{dh}}\right] \int_{-\infty}^{+\infty}\int \left|P_D(\vec{q})\right|^2 \times \exp\left[-\frac{ik_o n}{d_{dh}}\vec{q}\vec{\Delta}_\perp\right] d^2q dz \right|^2 \Big/ T^2 \times \int_{-\infty}^{+\infty}\int \left|P_D(\vec{q})\right|^2 d^2q. \quad (2)$$

Here the measured diffracted beam intensity $\mathbf{I_D(\Delta)}$ is normalized by its peak value at zero shift $\mathbf{I_{D(\Delta=0)}}$.

It follows from Eqn. (2) that any spatial mismatch between the hologram and reconstructing beam **R**(**r**) should result in a decline of the diffracted beam intensity. Figure 1 shows the fall-off in $I_{DN}(\Delta_\perp)$ that occurs for lateral shift $\Delta_\perp$. This figure for comparison includes also dependence $I_{DN}(\Delta_\perp)$ if calculated from a standard correlation function $I_{COR}(\Delta_\perp)$ from the statistical analysis of the speckle pattern. Comparison of these two curves clearly illustrates the impact of the spatial (volume) interaction on shift selectivity of the RER-beam hologram.

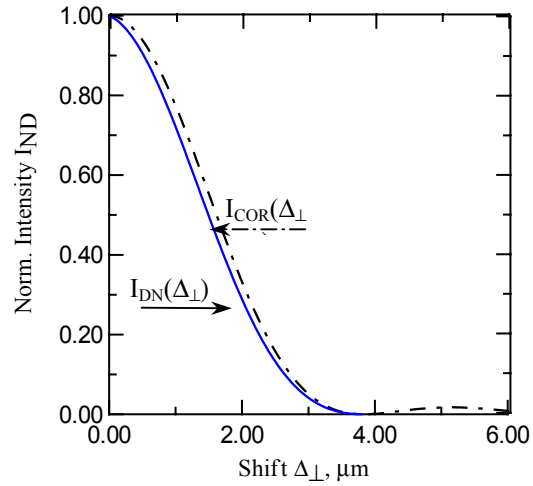


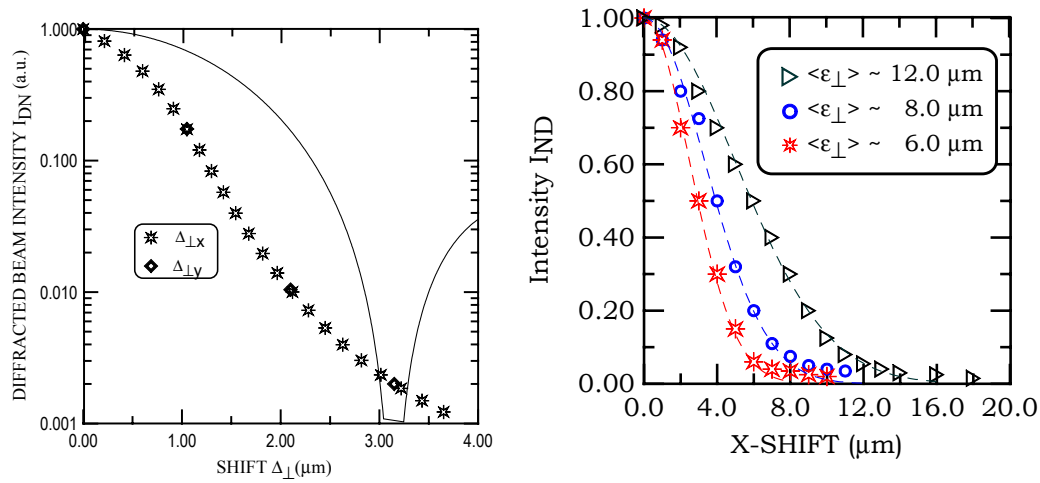
Figure 1. Diffraction beam intensity $I_{DN}(\Delta)$ as a function of lateral $\Delta_\perp$ shift.

3 Experimental Study

For experimental verification the RER-beam holograms were recorded in 2.3-mm-thick Fe:$LiNbO_3$ crystal. In a first set of the experiments the crystal was set onto an XY computer controlled positioning table (shift accuracy 0.025 μm in X-Y plane). A 1 cm diameter CW argon laser beam ($\lambda = 515$ nm, P = 40 mW/cm^2) was used as the coherent light source for hologram recording. The laser beam scattered by the ground glass diffuser is then picked up by a large aperture lens (f# = 1.4(forming a subjective speckle pattern in the recording plane. By changing the relative spatial position of the recording scheme elements allows for simple modification of average lateral speckle size $<\sigma_\perp>$. The RER-beam intersected with the plane wave signal beam at an angle of $\theta_o = 30^o$ in air ($\theta_{Ro} = 0^o$ and $\theta_{So} = 30^o$).

The diffraction efficiency of the hologram in its original position ($\Delta_\perp = 0$) was approximately 10^{-3}. After the hologram was recorded, a lateral shift $\Delta_\perp$ was introduced to evaluate the sensitivity of the reconstruction beam intensity upon lateral shift. A typical example of such operation is shown in Figure 2 for two orthogonal in plane (X-Y) shift direction (a) and for several values of the speckle size $<\sigma_\perp>$. The solid line in Figure 2 shows the behavior for the angular selectivity that would operate the diffracted beam intensity at identical conditions. Following data from Figure 2 the parameter of shift selectivity can be introduced for RER-beam hologram by analogy with angular selectivity of the plane wave hologram. It is evident also that speckle-shift selectivity has a very smooth character and contrary to the angular Bragg selectivity has no side-lobs in course

of displacement.



a) b)

Figure 2. Shift selectivity of RER-hologram (a) and its dependence upon average speckle size $<\sigma_{\perp}>$ (b).

3.1 Data recording-retrieval.

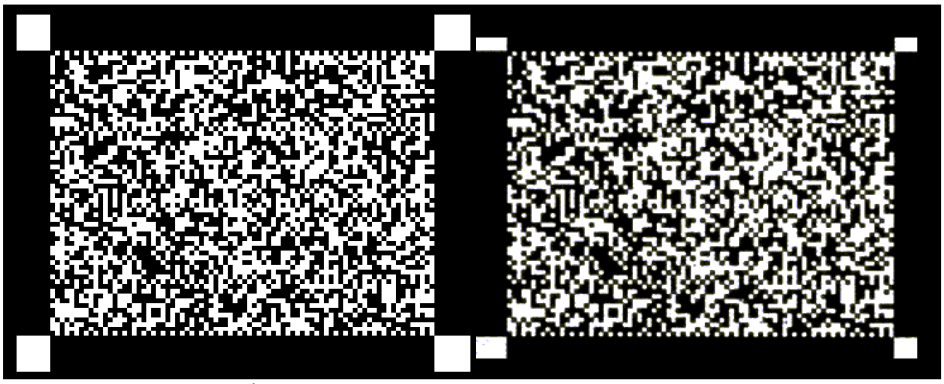
To verify experimentally the proposed data storage-retrieval concept, a breadboard system was constructed. A model GSL150/S CW diode-pumped Nd:YAG laser with output power 200 mW at λ = 530 nm was used as the light source. The SONY model LCX 026AL SLM with window size 2.3 x 2.3 mm and pixel size 22.5 x 22.5 μm was used to form a signal channel. The SLM was controlled by a PC that also had a National Instruments PCI-1407 single channel frame grabber for image retrieval. The data retrieval was arranged with the CMOS detector (pixel size 11.0 x 11.0 μm).

The detector location and limiting aperture were adjusted to produce the best SLM image onto a CMOS detector array. The pixel pitch of the CMOS detector was 12 μm x 12 μm, while that of the SLM was approximately 23 μm x 23 μm. The imaging optics was adjusted to produce a magnification of 1 SLM pixel to 2 CMOS pixels. Tests were also conducted using a CCD detector array with a pixel pitch of 8.4 μm x 9.8 μm in place of the CMOS. The magnification in this case was approximately 1 SLM pixel to 2.25 CCD pixels in one axis and 1 SLM pixel to 2.6 CCD pixels in the other.

Once the SLM and detector were properly aligned, tests were conducted in which the data area contained a known, random code and was projected onto either the CMOS or CCD detectors. Each bit of the code was represented by a value of either no attenuation or full attenuation over an area of the SLM. The exact scaling was calculated during each test by the program based on the location of the four dark corners generated by the SLM for alignment.

To test the reliability of the system using the initial test parameters, a series of known, random codes were written to the SLM and read back by the detector. For each test, the code read by the detector was converted back into a digital value and compared to the original code written to the SLM. Experiments in which several hundred codes were written and read were conducted and the location of bits that contained errors was

tracked. Typical results of the random code generation and optical read-out from the CMOS detector are shown in Figure 3, where both original (a) and retrieved (b) fields are presented.



a) b)

Figure 3. Original (a) and retrieved (b) data-page.

The system was found to be generally reliable; however, some data bits proved to be considerably more prone to errors than the rest. The system also seems to be extremely sensitive to slight changes in alignment. However, at correct alignment of the optical tract the performed tests with about 10^3 cycles allowed us to get the bit-error rate (BER) no higher than three for the entire field of the detection area.

3.2 RER-beam storage in reflection geometry.

As a part of the recording process optimization, we studied the possibility of data multiplexing using a reflection holography scheme. In this geometry the signal and reference beams are illuminating the recording medium from opposite directions, and in this way the reflection grating is formed. The essential benefit of the reflection geometry over the transmission one is the possibility of building a more compact memory module.

Experimentally study of the reflection mode geometry operation the speckle-encoded hologram was recorded with angle $\theta_R = 165^o$ between reference RER-beam and object beam. Average speckle size of RER-beam in this experiment was $<\sigma_\perp> \approx 7$ μm. As it was in transmission geometry the RER-beam was normal to the front surface of $LiNbO_3$ crystal, and C-axis (optical axis) of the crystal was normal to its front surface. The object beam had an incident angle 30°, propagating from the opposite direction.

No anomalies in the shift selectivity behavior have been observed in this geometry as compared to the transmission one, and a typically measured dependence of the normalized diffracted beam intensity upon spatial decorrelation between recorded and reconstruction positions of the RER-beam (shift selectivity) is shown in Figure 4. It is worth of noting at this point that the angular selectivity of the plane wave hologram recorded in a similar conditions was $\delta\Theta > 5^o$ that should result in extremely low storage density, while RER-beam selectivity provides a very good results.

4. Summary and Conclusion

In summary, the random encoded reference beam holographic recording demonstrates extremely high selectivity and therefore high data storage. This selectivity is based on the effects of spatial volumetric decorrelation between the recording and retrieving reconstruction field. Contrary to angular or spectral selectivity of the volume hologram, the two mechanisms that are traditional used for data multiplexing, the RER-beam holograms can be made free from *sinc*-type intensity modulation at its reconstruction. That should result in a much lover cross talk for this type of multiplexing. We demonstrated that the RER-beam hologram operates equally well in both transmission or reflection geometry. These features makes RER-beam hologram architecture attractive for building compact data storage system with ultra-high density and capacity.

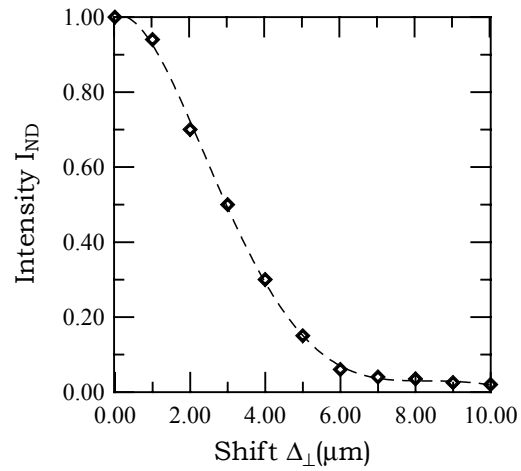


Figure 4. Shift selectivity in reflection geometry

References

[1] This research was sponsored in part by the US Department of Energy.

[2] Xin An, D. Psaltis, G. Burr, "Thermal fixing of 10,000 holograms in $LiNbO_3$:Fe," *Appl. Opt.*, **38**, pp. 386-393 (1999).

[3] J.Rosen, M. Segev, A. Yariv, "Wavelength-multiplexed computer generated holography", *Opt.Lett.*, **18**, pp.744-746, (1993).

[4] G. Rakuljic, V. Leyva, A. Yariv, "Optical data storage using orthogonal multiplexed holograms," *Opt.Lett.*, **17**, pp.1471(1992).

[5] S.Tao, D.Selvian, J.Midwinter, "Spatioangular multiplexed storage of 750 holograms in Fe:$LiNbO_3$ crystal", *Opt.Lett.*, **18**, pp. 912-914, (1993).

[6] Darskii, V. Markov, "Information capacity of holograms with a reference speckle-wave grating." *SPIE Proc.* **1600**, 318 (1992).

[7] V. Markov Yu. Denisyuk, R.Amezquita, "Speckle-shift hologram and its storage capacity", *Opt. Mem. Neural Net.* **6**, 91 (1997).

[8] V. Markov, M. Soskin, A. Khishnjak, V. Shishkov, Structural conversion of a coherent beam with a volume phase hologram in LiNbO3" *Soviet Tech. Phys. Lett.*, 4, pp.304-306, 1978.

[9] D. Psaltis, M. Leven, A. Pu, G. Barbasthtis, "Holographic storage using shift multiplexing," *Opt. Lett.*, **20**, pp. 782-784, (1995).

[10] A. Darskii, V. Markov, "Shift selectivity of holograms with reference speckle wave," *Opt.& Spectroscopy*, **65**, pp. 392-395,(1988)

iSCSI Initiator Design and Implementation Experience

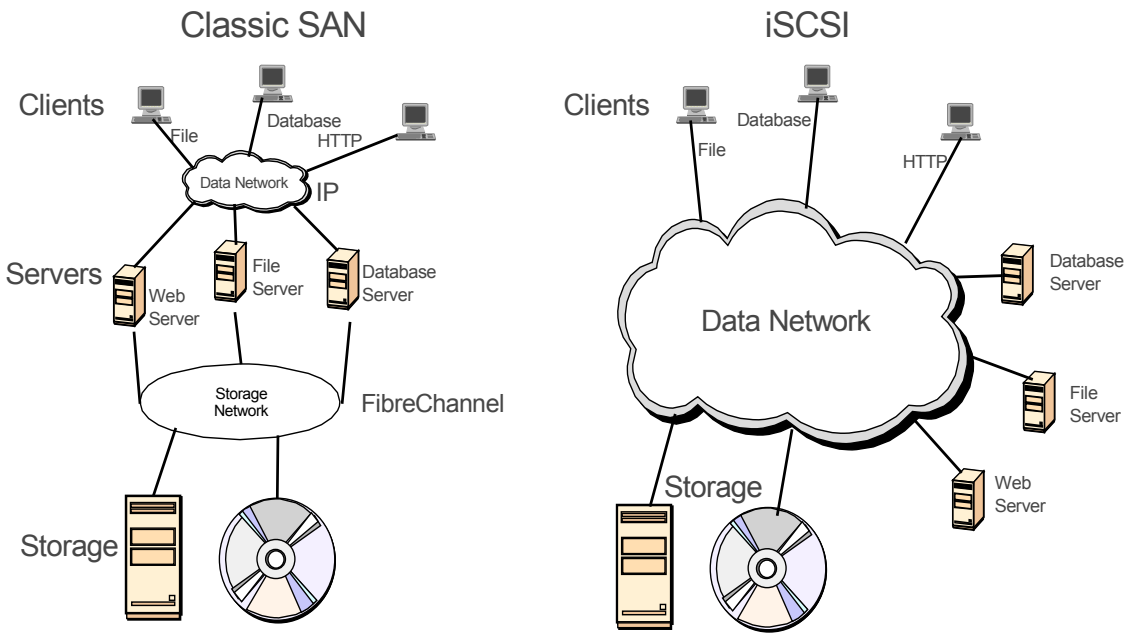
Kalman Z. Meth
IBM Haifa Research Lab
Haifa, Israel
meth@il.ibm.com
tel : +972-4-829-6341
fax: +972-4-829-6113

Abstract
The iSCSI protocol provides access to SCSI devices over a TCP transport. Using the iSCSI protocol enables one to build a Storage Area Network using standard Ethernet infrastructure and standard networking management tools. This paper outlines how we implemented a family of iSCSI initiators on a common core. The initially supported initiators were on the Windows NT and the Linux Operating Systems. Code for a version of the Linux iSCSI initiator has been released as Open Source. Initial testing indicates that iSCSI can provide reasonable performance relative to traditional storage environments.

1. Introduction

1.0 SANs and iSCSI

Storage Area Networks (SANs) provide a way for multiple hosts to access a shared pool of remote storage resources. Traditional SANs are built using FibreChannel technology [1] running the FCP protocol to carry SCSI commands over the FibreChannel network. Two separate network infrastructures are needed in an environment that uses a Local Area Network (LAN) for usual network activity and a SAN for shared remote: an Ethernet (or equivalent) infrastructure (running TCP and similar protocols) for usual LAN activity, and a FibreChannel infrastructure (running FCP protocol) for the SAN activity. iSCSI [2] is a protocol that carries SCSI commands over the TCP protocol [3]. iSCSI enables access to remote storage devices using TCP over standard LAN infrastructures. Using iSCSI dispenses with the need for a separate FibreChannel infrastructure and the need for a separate set of FibreChannel management tools. The difference between a traditional SAN and a possible iSCSI setup is depicted in the following figure.



1.1 iSCSI Overview

The iSCSI protocol [2] is a mapping of the SCSI remote procedure invocation model (see SAM2 [4]) over the TCP protocol [3]. Communication between the initiator and target occurs over one or more TCP connections. The TCP connections carry control messages, SCSI commands, parameters and data within iSCSI Protocol Data Units (iSCSI PDUs). The group of TCP connections that link an initiator with a target form a "session". The SCSI layer builds SCSI CDBs (Command Descriptor Blocks) and relays them with the remaining command execute parameters to the iSCSI layer. The iSCSI layer builds iSCSI PDUs and sends them over one of the session's TCP connections.

1.2 Design Goals

We designed and implemented a family of iSCSI initiators. Initial testing was performed against an IBM TotalStorage 200i disk controller using a standard 100Mbit Ethernet network connection. A major design goal of our initiators was to allow for multiple Operating Systems to work on the same common code base. Each operating system has its own set of interfaces for the SCSI subsystem and for its TCP transport. However, the implementation of the iSCSI specification should be common to all operating systems. When we upgrade to a different level of the iSCSI specification, only the common core needs to change, while the OS-dependent code should remain more or less intact.

Additional design considerations of our initiators included:

- Allow the initiator to simultaneously use devices from multiple target machines.
- Utilize multiple TCP connections between and iSCSI initiator and target.
- Utilize multiple processors if running on an SMP (Symmetric Multiprocessor).

1.3 Design Assumptions

The common core was designed and written using some basic assumptions about the Operating System (OS) on which it would be run. We assumed that the base Operating System would have the following features:

- Support for multiple threads in the kernel.
- Reading/writing from TCP can be easily abstracted into a single read/write function call.
- Some SCSI commands might be (re-) issued from inside the scsi completion routine, thus possibly running at some priority level for which we may not block.
- Task Management requests may arrive at some priority level, and hence we must provide an implementation that does not block, if requested.

We also had in mind a certain layering of the SCSI subsystem that seems to be prevalent in a number of Operating Systems. In both the Linux and Windows NT operating systems there are 3 layers to the SCSI subsystem. There is one high-level (class) driver for each type of SCSI device: disk, tape, CD, etc. There is a mid-level (port) driver that has common code for all types of devices, which takes care of command timeouts and serialization of commands. The low-level (miniport) driver is specific to an adapter, and must provide a queuecommand() or dispatch() routine that is used by the mid-level driver. This 3-level layering is essentially the model that is presented in the Common Access Method [5].

2. Implementation Details

2.1 Data Type and Function Abstractions

In order to build a common core, we abstracted the basic Operating System dependent data types and services that we would need to use, and defined these individually for each Operating System on which we implemented the iSCSI initiator driver. The basic data types that we defined are described here with their corresponding Linux (2.2) definitions.

```
typedef spinlock_t              iscsiLock_t;    /* basic spin lock */
typedef struct wait_queue*              iscsiEvent_t;   /* sleep event */
typedef uchar                   iscsiIrql_t;    /* interrupt level */
typedef struct scsi_cmnd                SCB_t;          /* SCSI Command Block */
typedef struct {                /* TCP connection descriptor */
        struct sockaddr_in      sin;
        struct socket           *sock;
        iscsiEvent_t            event;
} iscsiSock_t;
```

The basic services that must be provided by each Operating System and their corresponding Linux implementation are:

```
#define iscsiOSmalloc(size)   kmalloc(size, GFP_KERNEL)
#define iscsiOSlock(lock, irql)         spin_lock_irqsave(lock, (*irql))
```

```
#define iscsiOSunlock(lock, irql)      spin_unlock_irqrestore(lock, irql)
#define iscsiOSsleep(event)   sleep_on(event)
#define iscsiOSwakeup(event)          wake_up(event)
```

The common core uses these macros, which enable us to write code that is common to multiple platforms. The common core must also call some functions to perform TCP operations. Their prototypes are given below.

```
s32 iscsiOSreadFromConnection(iscsiSock_t *sock, void *buffer, u32 len, u32 offset);
s32 iscsiOSwriteToConnection(iscsiSock_t *sock, void *header, u32 headerLen, void *buffer, u32 len, u32 offset);
s32 iscsiOSmakeConnection(u32 addr, u16 portNum, iscsiSock_t *isock);
void iscsiOScompleteCommand(SCB_t *scb, u32 status);
```

The core provides a number of routines that the OS-dependent layer can call. The prototypes of the main core functions are given below.

```
s32 iscsiCoreCreateSession(u32 addr, u16 portNum, u32 nConnections, uchar *loginParams, u32 loginParamsLen);
s32 iscsiCoreEnterCmdInQ(SCB_t *scb, void *cdb, u32 cdbLen, u32 sessionhandle, iscsiLUN_t lun, void *data, u32 datalen, u32 flags);
u32 iscsiCoreResetDevice(u32 sessionHandle, SCB_t *scb, iscsiLUN_t lun);
```

The OS-dependent code calls iscsiCoreEnterCmdInQ() for each command that it wants to send to the target. The core then takes over and processes the command, sending it to the target, receiving a response from the target, and reporting the results back to the OS-dependent code by calling the iscsiOScompleteCommand() function.

2.2 Common Core General Structure

For each session established by the initiator (i.e. for each target), the initiator maintains a queue of items that must be sent to the target. We call this queue the command queue. The initiator also maintains a dedicated command queue handler thread to read items from this queue and to send them to the target.

The initiator maintains state of each command that has been sent to a target. This state information is saved in a table, indexed by an Initiator Task Tag. The target may send status or R2T (Ready to Transfer) PDUs to the initiator relating to a particular command. The Initiator Task Tag is used to easily look up the relevant information in the table.

For each TCP connection (even if we have multiple connections for a single session) the initiator maintains a dedicated thread to read from that TCP connection. The use of separate threads to read from each TCP connection and to send out messages allows us to better exploit the CPUs while waiting for data to arrive or be sent over a network connection. A read thread posts a read request on its TCP connection to receive an iSCSI header. The thread waits until data has arrived and has been placed in its buffer. The read thread parses the iSCSI header to determine how much additional data follows the

header. The read thread then posts a read request for the data of specified length, providing an appropriate buffer into which the data is to be placed. The read thread then performs whatever processing is needed to handle the PDU.

2.3 Implementation Lessons

In this section we briefly discuss some problems we encountered and lessons we learned in our implementation.

Windows NT expects command completion to occur from within an interrupt handler. Since we did not have any real hardware of our own, and all of our internal threads ran at regular priority, we broke a basic assumption of the Windows NT SCSI subsystem. We had to artificially create an interrupt in order to get the Windows NT SCSI subsystem to complete the processing of commands that were handled by our driver.

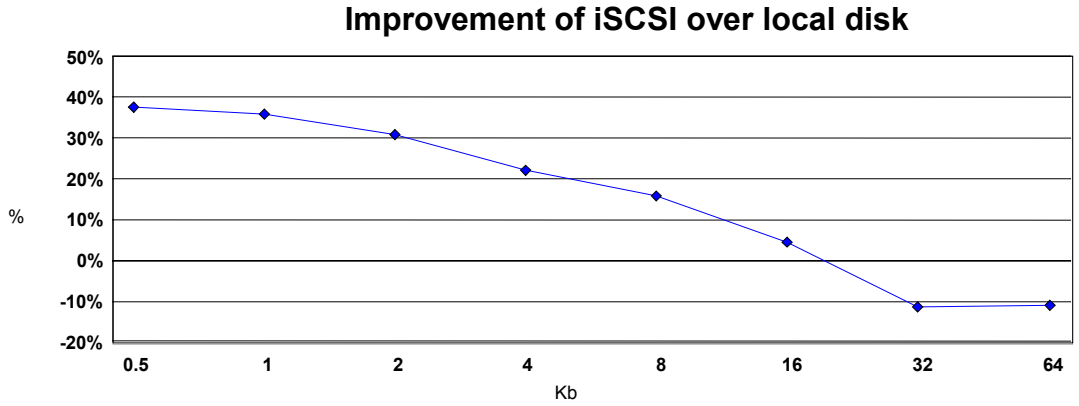
Linux also expects commands to be completed in a type of interrupt handler. A certain lock must be obtained and interrupts must be blocked when calling the Linux SCSI command completion handler.

In Linux, SCSI commands might be issued from within an interrupt handler. The call to iscsiCoreEnterCmdInQ() might therefore be called from within an interrupt handler, and any locks that we obtain in that routine must be safe to obtain and contend with an interrupt handler. It is therefore necessary to block interrupts whenever we obtain locks that may also be obtained at interrupt level inside the iscsiCoreEnterCmdInQ() function.

There is an inherent problem in mounting and unmounting iSCSI disks automatically upon reboot. In general, when the system first tries to mount its file systems, the network isn't yet up, thus preventing us from reaching our iSCSI disks. Also, the disk cannot automatically unmount cleanly during shutdown since by the time the system tries to sync its disks the network may already be gone.

2.4 Performance

We performed some preliminary measurements of iSCSI performance versus a directly attached IDE disk. We ran the IOTEST benchmark [7] on Linux for different sized block transfers and compared the results. The following graph shows the relative number of read operations between the iSCSI and local IDE disk.



For small data transfers iSCSI outperformed the local IDE disk by about 30%, despite the network overhead. This is due to the higher performance SCSI disks on the TotalStorage 200i target. Using the TotalStorage 200i SCSI disk locally outperformed iSCSI by about 3% for small transfers. For large data transfers, the network overhead started to take on a larger and larger impact, causing the iSCSI performance to be up to 12% worse than the local IDE disk. This is apparently due to the numerous interrupts needed for the packaging and processing of many small TCP packets for a large data transfer. This phenomenon should be alleviated when using Network Interface Cards (NICs) that offload the TCP processing, thereby reducing the number of interrupts that must be handled by the host.

3. Related Work

Network Storage is becoming more and more common, allowing remote hosts to more easily access remote and shared data. A number of studies have been performed that show that IP attached network storage can provide reasonable performance relative to FibreChannel and other storage environments. See, for example, [8] [9] [10].

Other early iSCSI initiator drivers have been made available as Open Source [6]. Some of these early implementations support a fixed target with a single TCP connection. Some of these implementations were written and tested for uniprocessors, and could not take advantage of the multiple processors in an SMP. Our implementation has the distinction of allowing multiple targets, multiple TCP connections within each session to a target, and the ability to fully exploit SMP machines. The performance achieved on a uniprocessor by the other software iSCSI initiator implementations that we tested (against the same target) were essentially the same as for our initiator.

4. Future Work

A version of our Linux initiator has been released as Open Source [11]. We continue to revise our driver to keep up with changes in the iSCSI specification as it evolves. Over time, we are adding additional features that are defined in the specification. We intend to perform comprehensive performance measurements and adjust our driver accordingly.

5. Conclusion

We implemented a family of iSCSI initiators utilizing a common core. We outlined our basic design objectives and how we implemented our initiators. When we upgraded to a newer version of the iSCSI specification, we were able to perform the necessary changes to the common core to correspond to the new iSCSI specification, while the OS-dependent parts of the code remained essentially unchanged. Our implementation allows multiple targets, multiple TCP connections within each session to a target, and the ability to exploit SMP machines. Using the IBM TotalStorage 200i target, iSCSI significantly outperforms the local IDE disk for small data transfers, but lags behind IDE for large data transfers, apparently due to the extra overhead of network interrupts.

Acknowledgement
The author has had the benefit of interacting and drawing on the experience, ideas, and help of many people that were involved in the iSCSI project. The author would especially like to thank Zvi Dubitzky, Eliot Salant and Liran Schour for their contributions.

References

[1] Benner, A. *Fibre Channel: Gigabit Communications and I/O for Computer Networks*, McGraw Hill, New York, 1996.
[2] Julian Satran, et al, iSCSI (Internet SCSI), *IETF draft-ietf-ips-iscsi-10.txt* (Jan 20, 2002); see www.ece.cmu.edu/~ips or www.haifa.il.ibm.com/satran/ips
[3] RFC793, Transmission Control Protocol, DARPA Internet Program, Protocol Specification, Sep 1981.
[4] SAM2, SCSI Architecture Model – 2, T10 Technical Committee NCITS (National Committee for Information Technology Standards), T10, Project 1157-D, Revision 20, 19 Sep 2001.
[5] CAM, Common Access Method – 3, draft of American National Standard of Accredited Standards Committee X3, X3T10, Project 990D, Revision 3, 16 Mar 1998.
[6] See www.ece.cmu.edu/~ips/IPS_Projects/ips_projects.html.
[7] See www.soliddata.com/products/iotest.html.
[8] Rodney Van Meter, Greg Finn, and Steve Hotz, "VISA: Netstation's Virtual Internet SCSI Adapter," in *Proceedings of the ACM 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (San Jose, Calif., Oct.), ACM Press, New York, 1998, 71-80. see also www.isi.edu/netstation.
[9] Wee Teck Ng, Hao Sun, Bruce Hillyer, Elizabeth Shriver, Eran Gabber, and Banu Ozden, "Obtaining High Performance for Storage Outsourcing", *FAST 2002, Conference on File and Storage Technologies*, Jan 2002.
[10] Garth A. Gibson and Rodney Van Meter, "Network Attached Storage Architecture", *Communications of the ACM*, Nov 2000, vol 43, no. 11.
[11] See oss.software.ibm.com/developerworks/projects/naslib.

Efficiently Scheduling Tape-resident Jobs*

Jing Shi, Chunxiao Xing, Lizhu Zhou
Department of Computer Science and Technology
Tsinghua University
Beijing 100084, P.R.China
Shijing@mails.tsinghua.edu.cn, {xingcx, dcszlz}@tsinghua.edu.cn
Tel: +86-10-62789150

Abstract
Many large-scale data-intensive applications need to use tape library to manage large data sets, thus it is critical to study the online access techniques of tape library. The focus of this paper is on efficient tape-resident jobs scheduling, which is the key technique for improving performance of tape storage systems. We present several scheduling algorithms for tape-resident jobs, discuss the effectiveness of scheduling policies under cache-limited and cache-unlimited condition, and show the results of simulation experiments.

1 Introduction

Many data repositories are expected to become huge, possibly counted by terabytes in size. Examples of such repositories include terabyte-level Telecommunications Call Detail Warehouse, petabyte-level Digital Libraries, exabyte-level National Medical Insurance Records, Zettabyte-level Spatial and Terrestrial Database and video and Audio Data Archives[1][2]. The management of such large data sets requires the use of tertiary storage, typically implemented by using tape libraries. As a result, accessing, analyzing, mining, and other data-intensive applications can comprise of many tape-resident jobs that retrieve either wholly, or in part, data from tapes.

Tape library is characterized by (1) the use of removable tape media and a robot arm, (2) sequential access of data, and (3) the performance bottleneck caused by tape access. Tape-resident job usually consists of more than one request, each of which must be completed before the job is finished, and uses disk cache space to store the data of its completed requests. To improve the performance of tape-resident jobs, we have to consider the following two problems -- the accessing latency of tape library, and the capacity limitation of disk cache for storing the retrieved data from tapes.

Previous studies mostly focus on the request scheduling of tape library to improve performance of robotic storage library[3][4][5][6]. But our goal is to schedule the jobs consisting of a set of requests to minimize the completion time of the whole job. A study closely related to ours is the one in which the scheduling problem of tape-resident jobs is reduced to well-known flow-shop scheduling[7]. However, it doesn't consider the optimal scheduling of tape libraries.

In this paper, we will introduce better scheduling strategies for executing tape-resident jobs. We will discuss how to improve the performance of tape-resident jobs by optimized

* This research is sponsored by the National Grand Fundamental Research 973 Program of China under Grant No.G1999032704

I/O performance of tape library, and discuss the effectiveness of scheduling policies under cache-limited condition or cache-unlimited condition by simulation study. Section 2 gives the scheduling problem description of tape-resident jobs. The scheduling algorithms will be presented in Section 3 and the simulation results for performance comparison of scheduling algorithms will be given in Section 4. Finally, Section 5 concludes the paper.

2 Problem Description

A tape-resident job consists of a set of requests, each of which is a read operation for a set of continuous blocks on a tape. We assume that the requests are independent of one another, that is, requests don't need to be executed in some forced order. The reason is that the access of tape library is much slower than that of disk, if processor begins to execute the job before the data involved in by its requests are all loaded into disk cache, then the job is possibly blocked for waiting unloaded requests. So we reduce the execution principle of tape-resident jobs to a simple form, that is, the job doesn't begin to execute until the data of its requests are all loaded into disk. This assumption means that the data of requests may be loaded by any order. The following Fig.1 is the description model of tape-resident jobs.

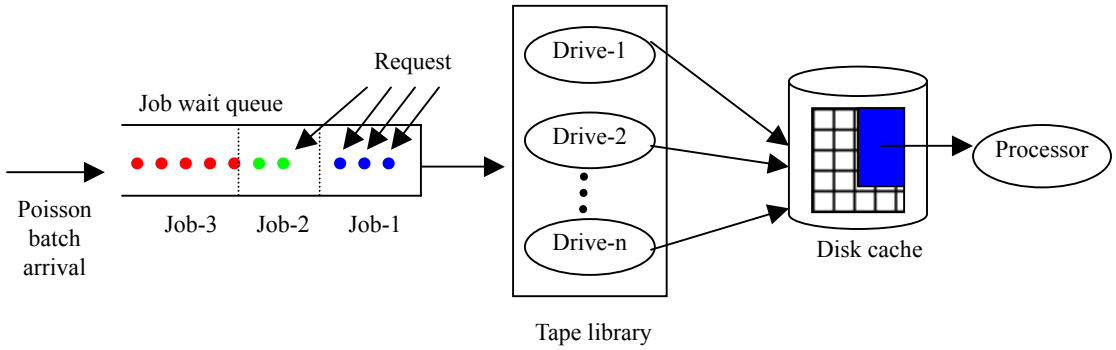


Fig. 1 The description model of tape-resident jobs

Since a job of several requests may involve more than one tapes, combining jobs that access the same media will make system process as much requests as possible in a tape schedule. One problem is that if jobs are not properly scheduled, the disk cache may be run out quickly. Therefore, it is critical to study the correlation between tape drive utilization and disk capacity limitation for tape-resident job scheduling. To do so, we consider the following optimization policies when designing tape-resident job scheduling algorithms:

- To improve the I/O performance of tape library
- To reduce resident time of data of jobs on disk cache
- To coordinate the input and output throughput of jobs to or from disk cache

3 Scheduling Algorithmic Issues

We study our scheduling problems under two kinds of restrictive conditions respectively: cache-limited and cache-unlimited. The former means the selection of scheduling policies must take the available space on disk cache into consideration, and the later assumes that

there is enough space of disk cache for scheduling. We first present five scheduling algorithms under the second condition, and then discuss these algorithms with the first condition of constraint. The algorithms focus on two key points: **tape selection policy**, and **scheduling list creation** (a scheduling list is an ordered list of requests for a selected tape).

(1) FCFS (First Come First Service). This algorithm services the jobs in the order of arrival, and always chooses the tape that the first request in job wait queue accesses to for next execution. The scheduling list of selected tape includes all requests that belong to the job and access the selected tape. These requests will be executed within one sweep of the tape.

(2) Max-EBW (Maximum Effective BandWidth). This policy improves the scheduling of tape-resident job in maximizing I/O performance of tape library. It always chooses the tape with maximum effective bandwidth for the next execution. The effective bandwidth of a tape is defined to be the total number of bytes transferred from the tape divided by the number of seconds consumed to perform this tape schedule.

(3) FCFS-PICKUP. This algorithm uses simplest tape selection policy--FCFS, namely, it always selects the tape to be accessed by the first request of a job in the wait queue, and then the algorithm inserts all requests of other jobs in the wait queue that will access the selected tape into its scheduling list, which is called the *PICKUP* policy for scheduling list creation.

(4) DYN-PICKUP. This algorithm has similar tape selection and scheduling list creation as FCFS-PICKUP. Besides this, it particularly considers the new arrival jobs. When the requests belonging to a new arrival jobs are trying to access the blocks on online tape that the tape head will pass over during the current sweep, they will be inserted into the running scheduling list. This is the dynamic policy for scheduling list creation.

(5) TUNING-PICKUP. This algorithm makes FCFS-PICKUP scheduling tunable. It uses *PICKUP intension factor F*, which indicates that PICKUP scheduling is only applied among the first *F* waiting jobs in the job wait queue, to tune the scale of scheduling list. Obviously, larger *F* means both larger cache occupation, and quicker response time. The selection of proper *F* value is the difficult point of this algorithm. Currently, we determine the *F* value by simulation experiments. A proper method for F value selection will be studied.

Above algorithms have different cache requirement: FCFS needs least cache space; TUNING-PICKUP may tune the size of cache occupation by changing PICKUP intension factor F; and other algorithms use more cache space than FCFS, but are not able to tune cache requirement. The comparison details of above algorithms will be given in next section.

4 Simulation Study

In this section, we give two groups of simulation results, each of which consists of two

figures: average response time of jobs and maximum cache requirement of jobs. The simulation parameters of tape library are based on Exabyte 220 tape library with two Eliant 820 drives and twenty EXABTYE 8mm tapes. In addition, we assume that the job arrival is stochastic and follows Poisson distribution. Each job averagely consists of 8 requests that have the average size of 64M bytes. We also assume that the disk cache should at least meet the maximum storage requirement of any job. The jobs are independent of one another.

Fig.1a and Fig.1b show response time and cache occupation curves for all algorithms except for TUNING-PICKUP. From the graphs we can observe that FCFS has least cache occupation but longest response time, and other algorithms significantly improve the average response time of tape-resident jobs by optimizing I/O performance of tape library. This performance improvement from tape library optimization has an associated cost in terms of storage space. The Figure also indicates that FCFS-PICKUP is the best scheduling policy. The reason is that it uses FCFS policy to speed up job output from disk cache while it takes advantage of PICKUP policy to improve I/O performance of tape library. Although the time performance of DYN-PICKUP policy is slightly better than that of FCFS-PICKUP, but its cache occupation is much higher than FCFS-PICKUP and Max-EBW. Its heavier workload creates proportionally larger storage requirement.

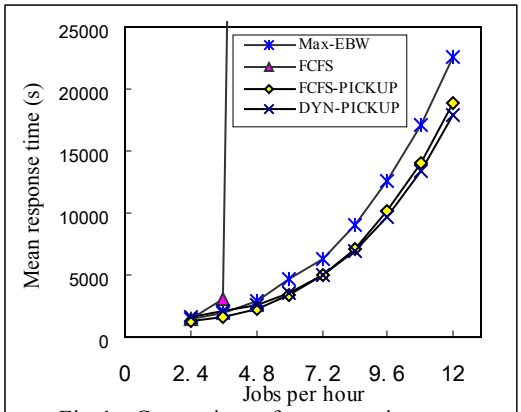


Fig. 1a Comparison of response time

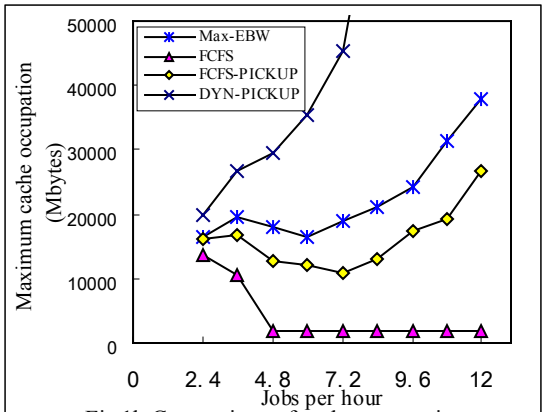


Fig.1b Comparison of cache occupation

The next simulation experiment explores the correlation between response time and cache space for FCFS-PICKUP algorithm and TUNING-PICKUP algorithm. We use *PICKUP intension factor F* to tune the size of cache occupation. This is very helpful in achieving a reasonable response time for tape-resident jobs when cache space is limited. Fig2a and Fig.2b illustrate when properly tuned, the time performance of TUNING-PICKUP is close to that of Max-EBW, but its space occupation is significantly reduced.

5 Conclusions

This paper discusses some efficient scheduling algorithms for tape-resident jobs. Our contributions include: (1) incorporate optimal I/O scheduling policies of tape library into the scheduling of tape-resident jobs so as to improve performance of tape-resident jobs by increasing the data throughput of tape library processing; (2) design better algorithm

FCFS-PICKUP for cache-unlimited system and TUNING-PICKUP for cache-limited system. The future work is to give a practical evaluation method for *PICKUP intension factor F* so that we may simply select *factor F* value for TUNING_PICKUP algorithm according to both workload and cache size.

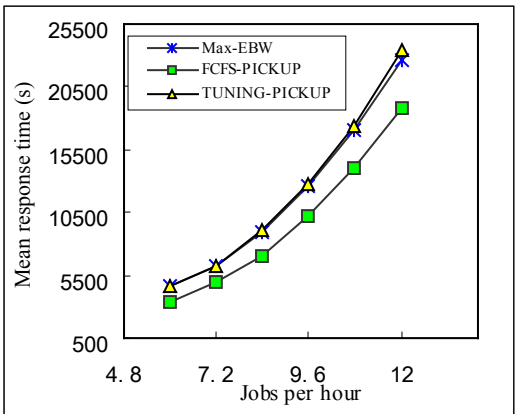


Fig. 2a The comparison of response time

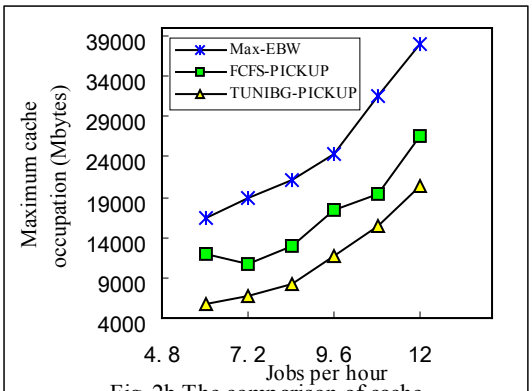


Fig. 2b The comparison of cache occupation

Reference

[1] Cariño F., Kaufmann A. and Kostamaa P., Are you ready for Yottabytes?, In Proc. of 17th IEEE symp. on Mass Storage Systems in Cooperation with the 8th NASA GSFC conf. on Mass Storage Systems and Technologies, pp. 476-485, Match 2000

[2] John Jensen, John Kinsfather and Parmesh Dwivedi. Data Volume Proliferation in the 21st Century--The Challenges Faced by the NOAA National Data Centers (NNDC), In Proc. of 17th IEEE symp. on Mass Storage Systems in Cooperation with the 8th NASA GSFC conf. on Mass Storage Systems and Technologies, pp. 335-350, Match 2000

[3] Bruce K.Hillyer and Avi Silberschatz， Random I/O Scheduling in Online Tertiary Storage Systems，In Proc. of the 1996 ACM SIGMOD Inter. Conf. on Management of Data, pp195-204, Canada, Jun 3-6 1996

[4] Bruce K. Hillyer, Rajeev Rastogi and Avi Silberschatz, Scheduling and Data Replication to Improve Tape Jukebox Performance, ICDE'99, pp. 532-541, 1999

[5] Toshihiro NEMOTO and Masaru KITSUEGAWA，Scalable Tape Archiver for Satellite Image Database and its Performance Analysis with Access Logs—Hot Declustering and Hot Replication--，In Proc. of 16th IEEE symp. on Mass Storage Systems in Cooperation with the 7th NASA GSFC conf. on Mass Storage Systems and Technologies, pp. 59-71, 1999

[6] Shi Jing and Zhou Lizhu, Dynamic Scheduling and Tuning to Improve Online Tape Library Performance, In Proceedings of the 6th International Conference for Younger Computer Scientists (ICYCS'2001), pages120-124, Oct. 2001

[7] Sachin More, S. Muthukrishnan and Elizabeth Shriver, Efficiently Sequencing Tape-resident Jobs, In Eighteenth ACM Symposium on Principles of Database Systems, 1999

The storage stability of metal particle media : Chemical analysis and kinetics of lubricant and binder hydrolysis

Kazuko Hanai ,Yutaka Kakuishi
Research & Development Center,Recording Media Products Div.,
Fuji Photo Film Co.Ltd.
2-12-1 Ohgi-cho
Odawara Kanagawa, 250-0001, Japan
hanai@mrdc.fujifilm.co.jp
kakuishi@mrdc.fujifilm.co.jp
tel: +81-465-32-2023
fax: +81-465-32-2170

Abstract

Archival life of MP (metal particles) tape is one of the biggest concerns for mass data storage users. The long-term stability of an MP tape is studied in terms of lubricant and binder systems. MP formulation tape that has been used for M2 videotape and DLT3 tape for more than fourteen years is analyzed. Gas chromatography (GC) and gel permeation chromatography (GPC) are used to analyze chemical changes of lubricant, fatty acid ester, and binder, polyester-polyurethane. The kinetics of hydrolysis of the fatty acid ester can be described by two first-order reactions. One is estimated to be corresponding to the hydrolysis of fatty acid ester on the surface of the magnetic layer, and the other to the fatty acid ester dissolved in the binder of magnetic layer. The hydrolysis of polyester-polyurethane (PU) can also be described by a first-order reaction. A durability test reveals that this MP tape keeps its good performance after long-term storage. A magnetization decrease of about twelve percent is observed after saving for fourteen years. This small decrease does not affect the above mentioned good performance.

1 Introduction

MP tape has been widely used in the fields of mass storage, broadcast, etc. In these fields, storage stability of MP tape is very important together with recording density. For development of MP media excellent in storage stability, it is necessary to know the problems in long-term storage. The claims during the use were investigated, and it became clear that many of them were due to the hydrolysis of the fatty acid ester as lubricant and the PU as binder.

As the first step of estimation of life expectancy of media, it was decided to study chemical changes of organic materials of MP formulation tape that has been used for M2 videotape and DLT3 tape for more than fourteen years. In addition, the magnetic properties and other physical characteristics were investigated and the durability was tested.

2 Experimental

The MP tapes for M2 stored in a laboratory for more than fourteen years were analyzed. Two types of fatty acid ester are contained as lubricants in the tape. One fatty acid ester is buthoxyethoxyethoxy stearate (BE2S) and the other is isoamyl stearate (AS). They were extracted with n-hexane from the tape and were quantified by GC (Shimadzu GC-17A).

Analysis of binder was also performed. Polyvinyl chloride and PU are contained as binder in the tape and they are crosslinked by hardener. PU used in a magnetic layer of this tape consists of methanediphenyl diisocyanate, hydroxycaproic acid, neopentyl glycol and phthalic acid. The magnetic layer was removed mechanically from magnetic tape and the soluble components were extracted with tetrahydrofuran. The extracts were analyzed by GPC (Toso HPLC8020) with an ultraviolet detector. Polyvinyl chloride shows no absorption in the ultraviolet region and only PU can be quantified.

3 Results and Discussion

3.1 Fatty acid ester

It was reported that lubricant loss in short term accelerating conditions is due to degradation and vaporization[1][2]. In our study, the decay of lubricant in long-term natural storage was investigated from a viewpoint of hydrolysis. It is known that the hydrolysis reaction of ester is first-order to ester concentration if enough water exists. The amount of fatty acid ester which remains in the tape and can be extracted with n-hexane is shown in Fig.1 and the decay rate constant of fatty acid ester is shown in Fig.2 and Tab.1.

The decay reactions of the two fatty acid esters are expressed as two first-order reactions of two steps in which each reaction rate differs. The ratio of reaction rate of BE2S to that of AS is shown in Tab.2. In the first step, although AS is smaller and more volatile than

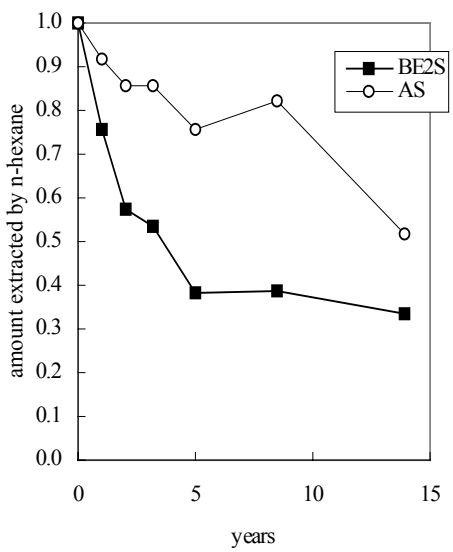


Fig.1 Decay of Fatty Acid Ester

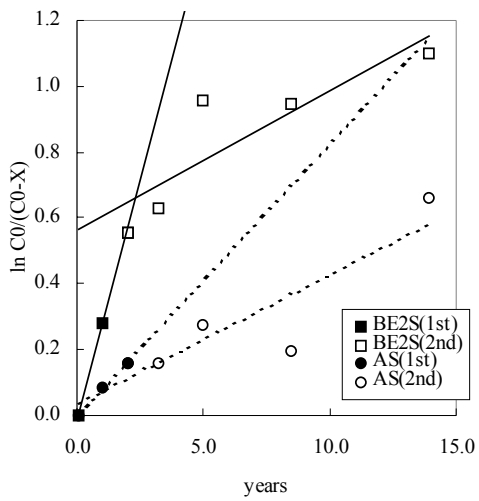


Fig.2 Reaction rate of Fatty Acid Ester
C0;initial amount
C0-X;remaining amount

BE2S, the decay loss reaction rate of BE2S is about 3.6 times of that of AS. This ratio of reaction rate is approximately equal to the ratio of hydrolysis reaction rate of fatty acid esters measured in acetone containing a small amount of HCl. Therefore it is considered that hydrolysis reaction is dominant in the first step. In the second step , the reaction rate of BE2S is about 1/6 of the first one and the difference of reaction rate of BE2S and AS is small, so vaporization is considered to be involved. It is assumed that the first decay

	/year	/sec
BE2S (1st)	2.9E-01	9.3E-09
BE2S (2nd)	4.3E-02	1.3E-09
AS (1st)	8.2E-02	2.6E-09
AS (2nd)	3.9E-02	1.2E-09

Table 1. The Decay Rate Constant of Fatty Acid Esters

1st	2nd	In acetone solution made weakly acidic with HCl
3.6	1.1	3.1

Table 2. The Decay Rate Ratio of BE2S/AS

comes from the fatty acid ester on the magnetic layer surface and the subsequent slow decay comes from the fatty acid ester dissolved in the magnetic layer binder. The thickness of the fatty acid ester layer on the surface is calculated to be about 0.7 nanometers from the quantity lost at the first step reaction and the surface area of the tape measured by gas-adsorption method. The concentration of the fatty acid ester dissolved in the magnetic layer binder is also calculated to be about 3 wt%, and is equal to the concentration at which binder films become opaque when the varying amount of fatty acid ester is added.

3.2 Polyester-polyurethane

/year	/sec
1.4E-03	4.3E-11

Table 3. The Decay Rate of Polyester Polyurethane

Though hydrolysis of PU of magnetic tapes had been reported [3][4], PU of MP tape in long-term natural storage was not yet investigated. We attempted to obtain the reaction rate of hydrolysis of PU using the ratio of the soluble component to PU content and the number average molecular weight of soluble PU. If one ester linkage of polymer is broken by hydrolysis, the number of PU molecules increases by one[5]. In order to find the hydrolysis reaction rate, the reciprocal of number average molecular weight was used

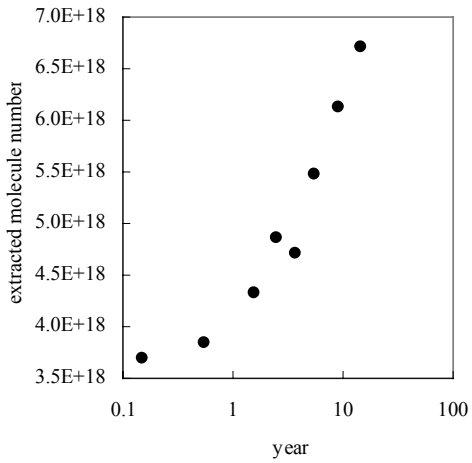


Fig.3 Molecule Number of Soluble PU
extracted from magnetic layer 1g

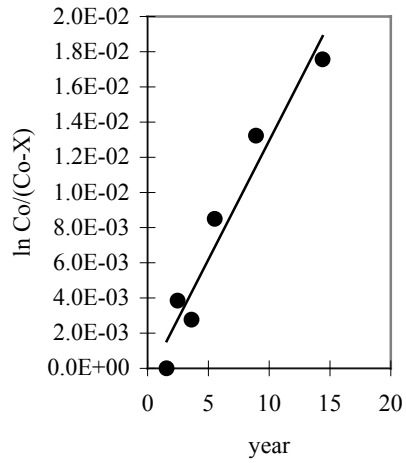


Fig.4 Reaction Rate of PU Degradation
Co;total linkage number in PU
X;ester linkage number broken
(=increased molecule number)

as the number of soluble PU molecules. The number of soluble PU molecules is shown in Fig.3. The number of soluble PU molecules increased remarkably after about 2 years, and hydrolysis reaction was considered to become predominant. Hydrolysis reaction rate was calculated using the increase of the number of molecules, as a first-order reaction shown in Fig.4 and Tab.3. The hydrolysis reaction of PU is extremely slow when compared to the hydrolysis of fatty acid ester in this magnetic tape. The reason is considered to be because PU is a high polymer and is crosslinked by hardener.

3.3 Physical characteristics and Durability test

Physical properties and video output level of a new tape and the tape stored for 14 years were measured and were compared in Tab.4. A 1-minute length x 1,000 passes running test using an M2 VCR was also performed as a durability test. Though remanence magnetization loss was about 12 % after 14 years, the decrease of video output level was 0.6 dB and was acceptable in practical use. The glass transition temperature of the magnetic layer did not change. The friction coefficient of the magnetic surface increased slightly but kept at a low value. After running for 1,000 passes as the durability test, the slight debris on the video heads was observed. But there was no difference between these two tapes in the amount of debris. These tests reveal that this MP tape keeps its good performance after long-term storage.

Storage Time (years)	0	14	Test Method
Magnetic Properties Br (Gauss)	2,640	2,320	VSM
Mechanical Properties Glass transition temperature of magnetic layer (degree at Celsius) Friction coefficient of magnetic surface	82 0.22	82 0.31	Dynamic viscoelastometer Vs. Stainless bar
Electro Magnetic Conversion Properties Video output (dB)	0	-0.6	M2 VCR

Table 4. Changes of Properties of MP Tape after Long-Term Storage

4 Conclusions

The physical characteristics and the chemical changes of the MP tape over 14 years were pursued, and the storage stability of the MP tape was proved to be satisfactory. The hydrolysis reactions of lubricant and binder in the MP tape could be expressed as first-order reactions, and the reaction rates were calculated. It becomes possible to make a quantitative comparison between the changes in the natural storage conditions and those in the accelerating tests. The thickness of surface fatty acid ester of the magnetic layer and the concentration of fatty acid ester dissolved in the binder can be estimated. Usually,these are very difficult to quantify by other techniques such as ESCA or AES because fatty acid esters are volatile and have no special element except carbon, hydrogen and oxygen in common as magnetic layer binder elements[6].

The accelerating conditions which can be used to simulate more precisely the passage of long time on the basis of these data will be established and be applied for the development of new media.

References

[1] The National Media Laboratory(NML) "Media Stability Studies Final Report,"p40(1994)

[2] E E Klaus,B Bhushan, "The Effects of Inhibitors and Contaminants on the Stability of Magnetic Tape Lubricants, " *Tribology Trans.,* **31** (1988) 276-281

[3] E F Cuddihy, "Aging of Magnetic Recording Tape, " *IEEE Trans. On Magn.,*MAG-**16** (1980) 558-568

[4] H N Bertram and E F Cuddihy, "Kinetics of the Humid Aging of Magnetic Recording Tape," *IEEE Trans. On Magn.,*MAG-**18** (1980), 993-999

[5] K.Yamamoto, "A Kinetic Study of Polyester Elastmer's Hydrolysis in Magnetic Tape , " *Proceedings of the 4th Sony Research Forum* (1995) 367-372

[6] M S Hemstock and J L Sullivan, " The Durability and Signal Performance of Metal Evaporated and Metal Particle Tape, " *IEEE Trans. On Magn.,***32** (1996) 3723-3725

Java and Real Time Storage Applications

Gary Mueller
195 Garnet St
Broomfield, CO 80020-2203
garymueller@qwest.net
Tel: +1-303-465-4279
Janet Borzuchowski
Storage Technology Corporation
2270 South 88th Street
M. S. 4272
Louisville CO 80028
janetborzuchowsk@qwest.net
Tel: +1-303-673-8297

Abstract
Storage systems have storage devices which run real time embedded software. Most storage devices use C and occasionally C++ to manage and control the storage device. Software for the storage device must meet the time and resource constraints of the storage device. The prevailing wisdom in the embedded world is that objects and in particular Java only work for simple problems and can not handle REAL problems, are too slow and can not handle time critical processing and are too big and can't fit in memory constrained systems.

Even though Java's roots are in the embedded application area, Java is more widely used in the desktop and enterprise environment. Use of Java in embedded real time environments where performance and size constraints rule is much less common.

Java vendors offer a dizzying array of options, products and choices for real time storage applications. Four main themes emerge when using Java in a real time storage application; compiling Java, executing Java with a software Java Virtual Machine (JVM), executing Java with a hardware JVM and replacing a real time operating system (RTOS) with a JVM.

The desktop and enterprise environment traditionally run Java using a software JVM that has been ported to a particular platform. The JVM runs as a task or process hosted by the platform operating system. With the performance and memory available on most workstations and personal computers, running an application on a software JVM is not an issue. However, many desktop and enterprise applications are not faced with the critical time and space constraints of an embedded application. Because of these constraints, running an embedded application on a software JVM incurs the additional overhead of software running software. Although it might be possible to run some embedded applications on a software JVM because of the tremendous speed of some processors, for most embedded applications, this configuration will not met timing or space constraints.

For a real-time storage application, running a JVM in software is typically only used for tasks which are not time critical. Typical tasks include hardware configuration,

maintenance and diagnostics, or upgrading or loading new code. For these tasks, a software JVM can meet the performance and space requirements. The software JVM typically runs as a low priority task. Other time critical tasks are written in C or C++ and do not use the intermediary JVM.

Compiled Java is an acceptable option since the JVM is eliminated and the functionality of the JVM such as garbage collection is wrapped into a set of runtime libraries. Compiling Java gives you the benefit of an object-oriented language without the performance penalty of an interpreted language.

The ultimate in speed and performance is attained when the JVM is cast in silicon. Several hardware vendors are planning or currently offering coprocessors or custom chips that execute Java directly in hardware.

Since the JVM provides the runtime environment for Java, in essence an operating system, one interesting approach is to use the JVM as a replacement for a RTOS.

This paper discusses the advantages and disadvantages of each approach as well as specific experiences of using Java in a commercial tape drive project.

1 Why Java for Real Time Storage Systems?

Java is an object-oriented language which gives you all the advantages of object technology, including faster delivery to market, more maintainable code, and easier adaptation to change. Java enforces the discipline of object design. Using Java in an embedded environment presents several challenges. Embedded applications have both functional and timing requirements and run in resource constrained environments. Java must meet the performance and space requirements of the embedded application. Some questions to answer include:

- Space the final frontier, will the JVM and class libraries fit?
- Performance, can the JVM run fast enough to meet hard real time deadlines?
- Scheduling, is the JVM deterministic and can non-deterministic tasks, such as garbage collection be scheduled?

2 Java Basics

Java is both a language and an environment which supports compilation and execution of the language.

Java, the language, supports single inheritance, polymorphism and other object concepts. Java is *compiled* to an intermediate language, Java byte codes, the assembly language for the JVM. The output of the Java compiler is a class file, which contains the Java bytecodes.

Java, the environment, is a virtual machine that has been ported to many operating systems and processors. The JVM interprets and executes the Java bytecodes and is usually written in C or C++. The JVM loads the Java class with a class loader, links the class files, verifies the bytes in a class file for correctness, prepares the class files for

execution, initializes the class, resolves method references and determines when to garbage collect unused classes. A typical Java environment is shown in Figure 1.

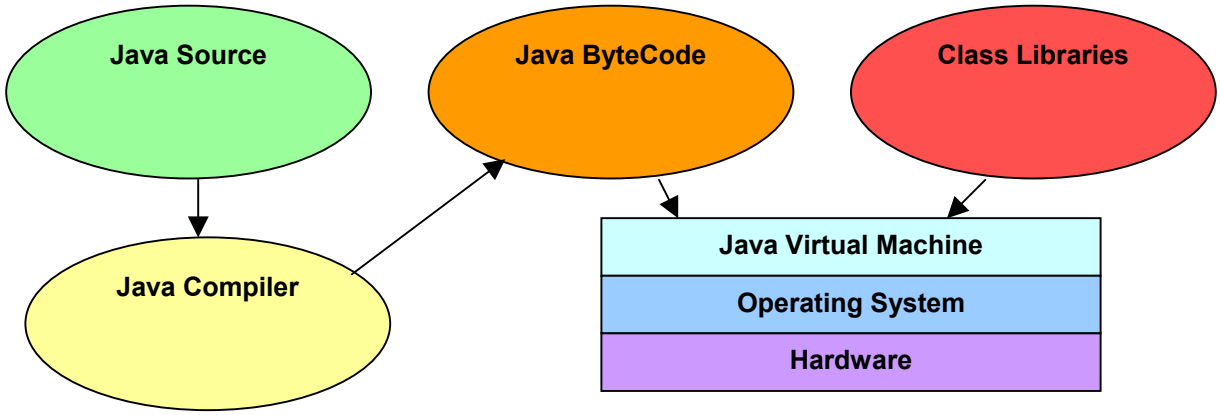


Figure 1 - Java Runtime Environment

3 Flavors or Java for Embedded Systems

There are four flavors of Java for embedded systems:

- Software Java Virtual Machine
- Compiled Java
- Hardware Java Virtual Machine
- Java as a Real Time Operating System

3.1 Software Java Virtual Machine

A software JVM is an application, process or task that typically is hosted by another operating system. Software JVMs are typically used for desktop or enterprise applications. Most desktop applications execute Java using a JVM running as a process or task on the desktop. Browsers execute Java with a JVM in the browser. This is the *classic* use of Java.

Since Java is interpreted by another program, the software JVM, there is a concern about the performance of the application which the JVM is executing. In particular, embedded applications must execute within specific time frames. Executing the embedded application on the JVM which itself is being executed raises the question of how fast the embedded application is executing and whether it can meet its required deadlines. One might speculate that there may exist embedded applications which given enough hardware horsepower will meet their required deadlines with a software JVM.

For those embedded applications which rely on and use a RTOS, a software JVM could be executed as a set of tasks or processes on the RTOS. Assuming the JVM tasks have a sufficient priority, some non real-time or slow real time embedded application tasks could be run using a software JVM such as:

- Hardware configuration

- Maintenance and diagnostics
- Code upgrades and loads

This method of executing Java is typical for desktop and enterprise applications where performance, although a concern, is not a driving factor. An example of this flavor of Java is WindRiver® Personal JWorks™ [1].

3.1.1 WindRiver® Personal JWorks™

As shown in Figure 2, Personal JWorks™ includes the PersonalJava Core Libraries, the JVM, the VxWorks Real Time Operating System (RTOS), the Supporting Native Libraries, a board support package (BSP) and device drivers for the particular processor and RTOS.

The PersonalJava Core Libraries include the applet, awt, beans, io, lang, math, net, rmi, security, sql, text and utl packages. The Personal JWorks™ application environment is based on the Java Development Kit 1.1.8 and adds security as specified in the Java 2 Software Development Kit, version 1.2.

Personal JWorks™ supports and fully implements the Abstract Windowing Toolkit (AWT) and fully supports the Java AWT graphics system. The WindRiver Media Library (WindML) glues the Personal JWorks™ environment to an applications graphics hardware. WindML supports 2D graphics primitives, fonts and provides audio and video support.

Personal JWorks™ uses a software JVM that runs as a set of tasks on VxWorks®. Using the Java Native Interface (JNI), JVM services such as thread and memory management (garbage collection), synchronization mechanisms, networking and graphics are mapped to VxWorks tasks through the Supporting Native Libraries. As a result, the VxWorks scheduler is able to prioritize and preempt the Java threads in the

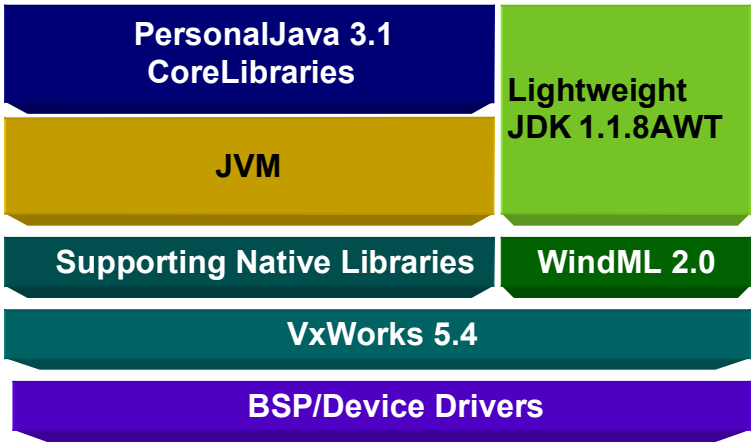


Figure 2- Personal JWorks™ Architecture

same way as it does VxWorks tasks. Although Personal JWorks™ does not provide real-time response, any VxWorks native task placed at a higher priority than a Java thread will execute without impact. Personal JWorks™ thus retains the determinism of VxWorks®. Using the JNI, Personal JWorks™ applications can access any C/C++ function in the VxWorks operating system including system calls.

3.2 Compiled Java

Compiled Java removes the environment portion of Java and treats Java as a language. Java is simply compiled to either native code or to an intermediate language such as C or C++. Compiled Java provides the benefit of an object-oriented language without the performance penalty of an interpreted language. Garbage collection and other JVM services are implemented through runtime libraries. Two examples of compiled Java are the Gnu Compiler for Java™ and WindRiver® Diab™ FastJ®.

3.2.1 Gnu Compiler for Java™(*gcj*)

Java applications are compiled and linked with the *gcj* runtime library, *libgcj*. The *libgcj* supplies the core classes, the garbage collector and the bytecode interpreter. The *libgcj* must be ported to the processor in your environment. The *gcj* allows three types of compiling:

- Java source code to native machine code
- Java source code to Java bytecode
- Java bytecode to native machine code

3.2.2 WindRiver® Diab™ FastJ®

FastJ® compiles C, C++ and Java source code to native machine code. As shown in Figure 3, the FastJ® compiler compiles, optimizes and generates assembly code for the desired target CPU and runtime environment using the Global Optimizer, Code Selector and Code Generator. External assembly source code and external libraries may be assembled and linked with the C, C++ and Java code. To reduce code size only needed core libraries may be configured. The Assembler together with the Linker produce an ELF format executable image for the desired processor.

FastJ® supports three memory management options:

- Explicit memory management, similar to C/C++, eliminates garbage collection.
- Standard, non-incremental garbage collection, runs when memory is low or explicitly called.
- Preemptive, incremental garbage collection, runs as a preemptable, low priority background task.

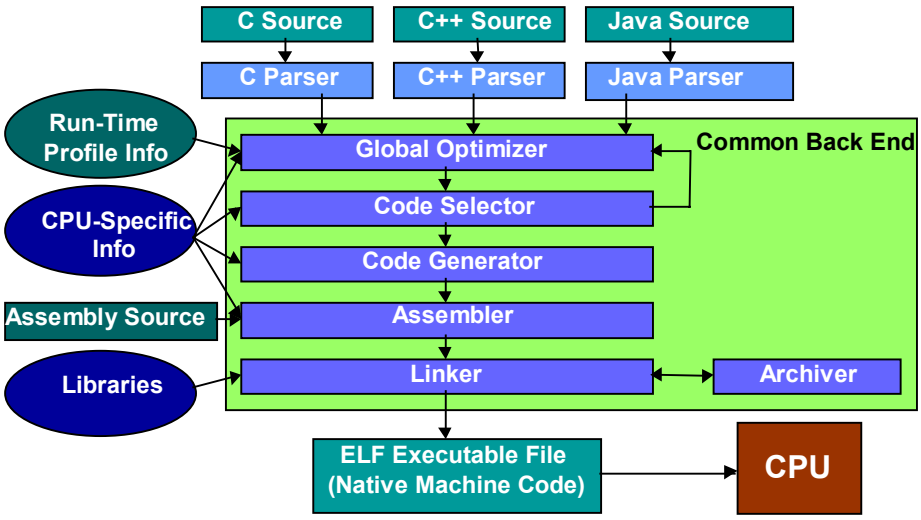


Figure 3 - FastJ® Compiler Architecture

3.3 Hardware JVM

The ultimate in performance is achieved by executing or running the JVM in hardware. The JVM is implemented in silicon as either a co-processor or separate processor on a custom chip. Specially designed or custom hardware is required which directly executes the Java bytecodes. This is similar to assembly code being executed on a particular processor. Several chip vendors including ARM from England, Ajile from the United States, Vulcan Machines LTD from England and NTT Docomo from Japan offer hardware JVMs. [2]

Several variations of the hardware theme are currently available. Some hardware implementations use a co-processor to execute Java bytecodes. Other implementations use specialized hardware, which is called when Java bytecodes are detected. An example of a hardware JVM is the ARM® Jazelle™[3].

3.3.1 ARM® Jazelle™

Jazelle™ is a product from ARM®, which includes a hardware JVM for the ARM® family of processors and a runtime environment to support Java applications. The Jazelle™ runtime architecture, as shown in Figure 4, allows Java applications to access the Java Class libraries available in the particular Java development kit, either the Java 2 Enterprise, Standard or Micro Edition. Each edition of Java has a virtual machine which executes the Java bytecodes. Jazelle™ currently supports the pJava, KVM and CVM virtual machines. Jazelle™ provides a Java Technology Enabling Kit for porting other VM's.

The Jazelle™ Supporting Code replaces the Java virtual machine interpreter loop and enables execution of the Java bytecodes directly in hardware. A condition bit in a new ARM® instruction puts the processor in the Java state. The processor then executes the Java byte code directly in hardware. Jazelle™ supports execution of both Java bytecodes and ARM® machine codes. This allows existing application written in C and C++ to continue to execute alongside the Java applications. The main difference between a software JVM such as Personal JWorks™ and a hardware JVM such as Jazelle™ is how the Java bytecodes are executed. In Personal JWorks™, the bytecodes are translated to native machine code and then executed. With Jazelle™, the bytecodes are executed directly in hardware.

Since the JVM must be supported by the underlying RTOS, Jazelle™ also supports WindowsCE, SymbianOS, PalmOS, Linux and many real time and proprietary operating systems.

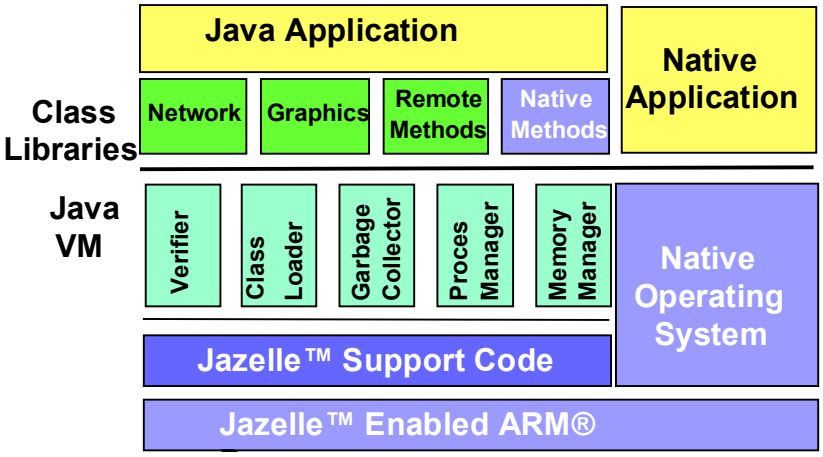


Figure 4 - Jazelle™ Run-Time Architecture

3.4 Java as a Real Time Operating System

An interesting variation is viewing the JVM as an operating system. The JVM is the RTOS. Since the JVM is essentially a machine, simulated or executed on another machine, it makes sense to eliminate the other machine and execute the JVM directly on hardware. An example of this is Jbed™ from Esmertec [4].

3.4.1 Esmertec ™

Jbed™ combines the JVM and a real time operating system into a single entity. Jbed™ has a four layer architecture. The Java applications have access to lang, io, util as well as the connection framework in the javax.microedition package and is PersonalJava 3.0 (JDK 1.1) compliant. As shown in Figure 5, Jbed™ supports many of the popular

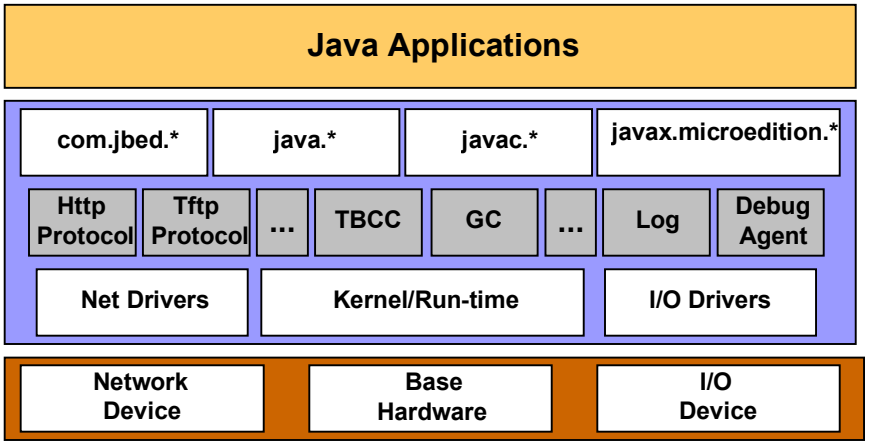


Figure 5 - Jbed™ Run-Time Architecture

Internet protocols such as HTTP, TFTP, TCP/IP, PPP and UDP. JVM services such as garbage collection (GC) are supported without the intermediary JVM. Jbed™ does not execute or interpret Java bytecode. Instead, bytecode is translated into fast machine code prior to downloading or upon class loading with the Way Ahead of Time compiler and the Target Bytecode Compiler (TBCC). This avoids the speed and size penalty of a JVM, yet stills provides advanced Java features such as dynamic code loading and automatic garbage collection. Jbed™ extends the Java thread package to provide priority based scheduling using the earliest deadline first algorithm. A device driver support package supports driver development in Java. Thus, the entire application including device drivers can be written in Java.

4 On the Road to Java

The 9840 and 9940 family of StorageTek tape drives use an ARM7® 32 bit processor, with 2-4MB of RAM for loading the code image. A 32MB - 64MB data buffer is used for data transfer and the drives support the SCSI, ESCON, and Fibre Channel interfaces. Specialized Application Specific Integrated Circuits (ASICs) are used to control the tape drive. All of the code is written in C with Vertex serving as the RTOS.

C++ and object design have been introduced into the time critical tape microcode. Initially, the classes have been written in C++ and are mirrored in Java for unit testing. The Java classes form the basis for a hardware simulator.

Since FastJ™ is similar to current development environment, FastJ™ will be the first step to introducing Java in our real time system. It is the least disruptive and does not require hardware changes. FastJ will be used to compile the Java classes used in the hardware simulator and tape microcode. Since the current RTOS is old, the next step will be to investigate Jbed™ which is a Java RTOS, a combination of hardware/software. Finally, since Jazelle™ requires hardware changes, the last step will be Jazelle™.

5 Summary

Recently, there has been a resurgence in the use of Java for embedded systems. Options ranging from software Java Virtual Machines offered by real time operating system vendors to chip vendors developing Java chips are available to the embedded storage developer. Java will be used in the next generation Personal Digital Assistants (PDA), such as the Palm Pilot, and in the next generation of mobile phones.

We believe that Java has now become a viable option for building real-time storage applications. Issues involving the space, performance and scheduling problems of Java for embedded systems are being solved. Almost daily, a new vendor or company announces its plan for Java in the embedded environment. With the many options available, at least one flavor of embedded Java will work for your application.

6 References

[1] WindRiver web site - http://www.windriver.com

[2] EE Times, January 29,2001, “Java Vendors set to skirmish over cellular”, page 1

[3] EE Times, October 16, 2001, “ARM tweaks CPU schemes to run Java”, page 20

[4] JavaPro, February, 2002, “A Comfortable Jbed”, page 72

DIR-2000, 1 Gbit/sec Data Recorder for VERA Project

Tony Sasanuma, Ph.D.
Sony Broadband Solutions Network Company
4-14-1 Asahi-cho, Atsugi-shi,
Kanagawa, 243-0014 Japan
Tony.Sasanuma@jp.sony.com
Tel: +81-46-230-5364
Fax: +81-46-230-6075

Abstract
This paper will discuss the new technologies used in the DIR-2000, 1 Gbit/sec data recorder: the highest performance in the commercial market. It will briefly explain how the DIR-2000 is implemented in VERA Program [1] of National Astronomical Observatory in Japan.

1 Introduction

More than 1000 units of Sony DIR-1000 Series [2] data recorders are being used for the varieties of applications among government and scientific communities worldwide. Responding to the request of a higher data rate than 512 Mbit/sec, Sony developed the DIR-2000 that offers the highest data rate of 1 Gbit/sec. The data capacity on 19mm metal particle tape is 600 GB and the recording time per cassette tape is 80 minutes at the data rate of 1 Gbit/sec.

2 New Format

Since 1990, ANSI ID-1 19 mm Format has been well accepted as the high performance and reliable format by variety of data recorder communities, and there are many ID-1 users worldwide. However, the data capacity per tape of 100 GB for ID-1 is not enough for a 1 Gbit/sec recorder, since the recording time would be only 13 minutes.

Sony is preparing to propose a new 19 mm format in ANSI Committee for standardization. The new format of 19 mm is not only suitable for data recording of high performance and high reliability demanded in 21st century, but also for read compatibility of ID-1 tape and similar interface and control on ID-1 drives. The specifications and parameters of the DIR-2000/new format are shown in terms of the comparison with the DIR-1000H / ID-1 Format in Table 1.

The dimensions of the DIR-2000 are the same as those of the DIR-1000 Series, so that they can be installed in the existing Sony' Mass Storage System such as PetaSite DMS-8800 and the DMS-24.

	DIR-1000H	DIR-2000
Format	ANSI ID-1 Format	New Format. ID-1 Read Compatible
Data Rates	512, 400, 256 Mbit/sec	1024, 512, 256 Mbit/sec
Data Capacity/Tape	100 GBytes	600 GBytes
Recording Time	25 minutes at 512 Mbps	80 minutes at 1024 Mbps
Media	Co-oxide	New Metal Particle
Tape Width	19 mm	19 mm
Tape Thickness	16 μm	11 μm
Coercive Force (Hc)	900 Oe	2300 Oe
Shortest wavelength	0.89 μm	0.45 μm
Track Pitch	45 μm	19 μm
Maximum Tape Speed	847.5 mm/sec	356.6 mm/sec
Recording Bit Rate/Head	88 Mbps	88 Mbps
Record / Playback Heads	16 heads/16 heads	32 heads/32 heads
Processor Channels	8 channels	16 channels
Maximum Writing Speed	39.5 meter/sec	19.7 meter/sec
Scanner Rotation Speed	110 rps at 512 Mbps	55 rps at 1024 Mbps
Data Interface	ECL Parallel with clock	
Control Interface	RS-422/IEEE-488GPIB/RS-232C	
Dimensions (W x H x D)	436 x 432.5 x 633.5 mm (17 1/4 x 17 1/8 x 25 1/8 inches)	
Weight	64 Kg (141 lb 1 oz)	70 Kg (154 lb 5 oz)
Power Requirement	100 V to 240 V AC ±10% (50/60 Hz)	
Power Consumption	800 VA	850 VA

Table 1. Specifications and Parameters

3 New Technologies

In order to meet the requirements of high data rate, high data capacity, long head life, less tape damage, and backward compatibility all together, new heads and new tapes were developed and implemented in new recorders.

3-1 Ferrite Cover over Heads and ETF Record Heads

There are 32 record heads and 32 playback heads: the total of 64 heads on the scanner of the DIR-2000! Since the spacing between a record head and playback head is small, the cross feed signal from record heads to playback heads would be significant during read-after-write that is an essential function for reliable data recording.

We introduced patented ferrite covers over record and playback heads to shield the magnetic flux. This simple idea of shielding is very effective and improves the cross feed by 12 dB.

The newly developed the ETF (Embedded Thin Film) head has small magnetic core compare with the conventional MIG (Metal In Gap) head, so that the magnetic leakage flux from record head is improved further by 7 dB.

3-2 Laminated Amorphous Playback Heads and New Metal Tape

The shortest wavelength becomes one half of ID-1 (0.45 μ m vs. 0.89 μ m), and the track pitch becomes less than one half of ID-1 (19 μ m vs. 45 μ m). In spite of these reductions, the combination of the laminated amorphous playback heads and newly developed metal particle tape provide even better C/N than ID-1 recorder. This could be achieved by the joint R & D of heads/drums, drives, and media in Sony.

3-3 Trench Design Heads

There are two trenches on record and playback heads of the DIR-2000. This patented head design provides better head-tape contact with lower head projection, larger head contour, and lower tape tension. These result in longer head life and less tape damage. Backward compatibility of format requires playback of tapes of different thickness. Trench design heads provide a good head-tape contact for different kinds of tape thickness throughout head life.

The same technologies of trench ETF/amorphous heads and new metal tapes are used in Sony Computer Tape Drive DTF-2 (24 Mbytes/sec via SCSI or Fiber Channel) that are installed as a few hundred TB Systems at NASDA and ERSDAC in Japan.

4 Applications

The first application for the DIR-2000 was VERA Project of National Astronomical Observatory in Japan. VERA stands for VLBI (Very Long Baseline Interferometer) Exploration of Ratio Astronomy. VERA array consists of four telescopes whose diameter is 20 meters (67 feet). The combination of these telescopes can obtain the resolution power of a telescope whose diameter is 2000km (1250 miles).

The DIR-2000 1 Gbit/sec recorder is one of the key devices for VERA Project. One DIR-2000 drive is used to record the data at each of four VERA telescope stations. The correlator at National Astronomical Observatory in Tokyo supports four tape drives of the DIR-2000 to analyze the data from four telescope stations.

The DIR-2000's are installed in the DMS-24, Mass Storage System for automated operations for data acquisition at the telescope stations and correlation in Tokyo. The DMS-24 library can handle up to 24 large cassette tapes (14.4 TB capacity) and two drives of the DIR-2000's.

Besides VERA Project, a government agency in Japan plans to develop 2.5 Gbit/sec ATM network, and is considering using the DIR-2000 to record the data on the broadband network. Broadband network is one of the important technologies in 21st century, and recording of high-speed un-interrupted data will be needed.

5 Conclusions

Sony has developed the DIR-2000: 1 Gbit/sec data recorder with 600 GB data capacity per tape. The DIR-2000 meets the requirement for recording of un-interrupting data at very high data rate. The applications for this recorder are not only scientific researches but also broadband radar and network.

The DIR-2000 will be demonstrated at Vendor Exhibit Area.

References

[1] M. Homma, et al. "Science with VERA: VLBI Exploration of Radio Astrometry" SPIE Proceeding No. 4015, 2000

[2] T. Sasanuma. "New 512 Mbit/sec ID-1 Recorder" THIC Conference October 15th, 1996

Index of Authors

Adam, Nabil R - Efficient Storage and Management of Environmental Information 165
Atluri, Vijayalakshmi - Efficient Storage and Management of Environmental Information 165
Azagury, Alain - Point-in-Time Copy: Yesterday, Today and Tomorrow 259

Banachowski, Scott A - Intra-file Security for a Distributed File System 153
Bhide, Anupam - File Virtualization with DirectNFS 43
Borzuchowski, Janet - Java and Real Time Storage Applications 317
Brandt, Scott A - Intra-file Security for a Distributed File System 153
Burns, Randal - Experimentally Evaluating in-place Delta Reconstruction 137
Butler, Michelle L. - Storage Issues at NCSA: How to get file systems going wide and fast within and out of large scale Linux cluster systems 93

Calvo, Sherri - Conceptual Study of Intelligent Data Archives of the Future 75
Chang, Tai-Sheng - Efficient RAID Disk Scheduling on Smart Disks 121

Debiez, Jacques - High Performance RAIT 65
Dee, Richard H - The Challenges of Magnetic Recording on Tape for Data Storage (The One Terabyte Cartridge and Beyond) 109
Du, David H C - Efficient RAID Disk Scheduling on Smart Disks 121

Engineer, Anu - File Virtualization with DirectNFS 43

Factor, Michael E - Point-in-Time Copy: Yesterday, Today and Tomorrow 259
Fitzgerald, Keith - Storage Area Networks and the High Performance Storage System 225

Graf, Otis - Storage Area Networks and the High Performance Storage System 225
Gu, Junmin - Storage Resource Managers: Middleware Components for Grid Storage 209

Hanai, Kazuko - The storage stability of metal particle media : Chemical analysis and kinetics of lubricant and binder hydrolysis 311
Harberts, Robert - Conceptual Study of Intelligent Data Archives of the Future 75
Hersch, Roger D - Indexing and selection of data items in huge data sets by constructing and accessing tag collections 181
Hughes, James - High Performance RAIT 65
Hulen, Harry - Storage Area Networks and the High Performance Storage System 225

Kakuishi, Yutaka - The storage stability of metal particle media : Chemical analysis and kinetics of lubricant and binder hydrolysis 311
Kanetkar, Anshuman - File Virtualization with DirectNFS 43
Karamanolis, Christos - File Virtualization with DirectNFS 43
Karamanolis, Christos - Locating Logical Volumes in Large-Scale Networks 271
Karlsson, Magnus - Locating Logical Volumes in Large-Scale Networks 271
Kempler, Steve - Conceptual Study of Intelligent Data Archives of the Future 75
Khizroev, Sakhrat - Perpendicular Recording: A Future Technology or a Temporary Solution 1

Khoo, Patrick Beng T - Introducing A Flexible Data Transport Protocol for Network Storage Applications 241
Kiang, Richard - Conceptual Study of Intelligent Data Archives of the Future 75
Kini, Aditya - File Virtualization with DirectNFS 43

Li, Jiangtao - Data Placement for Tertiary Storage 193
Litvinov, Dmitri -Perpendicular Recording: A Future Technology or a Temporary Solution 1
Long, Darrel D E - Experimentally Evaluating in-place Delta Reconstruction 137
Lynnes, Chris - Conceptual Study of Intelligent Data Archives of the Future 75

Mahalingam, Mallik - Locating Logical Volumes in Large-Scale Networks 271
Markov, Vladimir B. - High-density holographic data storage with random encoded reference beam 291
McConaughy, Gail - Conceptual Study of Intelligent Data Archives of the Future 75
McDonald, Ken - Conceptual Study of Intelligent Data Archives of the Future 75
Meth, Kalman Z - iSCSI Initiator Design and Implementation Experience 297
Micka, William - Point-in-Time Copy: Yesterday, Today and Tomorrow 259
Miller, Ethan L - Intra-file Security for a Distributed File System 153
Milligan, Charles - High Performance RAIT 65
Mueller, Gary - Java and Real Time Storage Applications 317
Muntz, Dan - Building a Single Distributed File System from Many NFS Server -or- The Poor-Man's Cluster Server 60
Muntz, Dan - File Virtualization with DirectNFS 43

Peterson, Zachary N J - Intra-file Security for a Distributed File System 53
Prabhakar, Sunil - Data Placement for Tertiary Storage 193
Ponce, Sebastien - Indexing and selection of data items in huge data sets by constructing and accessing tag collections 181

Ramapriyan, H. K - Conceptual Study of Intelligent Data Archives of the Future 75
Roelofs, Larry - Conceptual Study of Intelligent Data Archives of the Future 75
Ruwart, Thomas M - OSD: A Tutorial on Object Storage Devices 21

Sarkar, Prasenjit - IP Storage: The Challenge Ahead 35
Sasanuma, Tony - DIR-2000, 1 Gbit/sec Data Recorder for VERA Project 327
Satran, Julian - Point-in-Time Copy: Yesterday, Today and Tomorrow 259
Shi, Jing - Efficiently Scheduling Tape-resident Jobs 305
Shoshani, Arie - Storage Resource Managers: Middleware Components for Grid Storage 209
Sim, Alex - Storage Resource Managers: Middleware Components for Grid Storage 209
Stockmeyer, Larry - Experimentally Evaluating in-place Delta Reconstruction 137
Sun, Donglian - Conceptual Study of Intelligent Data Archives of the Future 75

Thunquest, Gary - File Virtualization with DirectNFS 43

Vila, Pere Mato - Indexing and selection of data items in huge data sets by constructing and accessing tag collections 181
Voruganti, Kaladhar - IP Storage: The Challenge Ahead 35

Wang, Wilson Yong H. - Introducing A Flexible Data Transport Protocol for Network Storage Applications 241
Watson, Richard W - Storage Area Networks and the High Performance Storage System 225

Xing, Chunxiao - Efficiently Scheduling Tape-resident Jobs 305
Xu, Zhichen - Locating Logical Volumes in Large-Scale Networks 271

Yesha, Yelena - Efficient Storage and Management of Environmental Information 165
Yu, Songmei - Efficient Storage and Management of Environmental Information 165

Zhang, Zheng - File Virtualization with DirectNFS 43
Zhou, Lizhu - Efficiently Scheduling Tape-resident Jobs 305

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
April 2002

3. REPORT TYPE AND DATES COVERED
Conference Publication

4. TITLE AND SUBTITLE
Tenth Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Nineteenth IEEE Symposium on Mass Storage Systems

5. FUNDING NUMBERS
423

6. AUTHOR(S)
Benjamin Kobler and P C Hariharan, Editors

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS (ES)
Goddard Space Flight Center
Greenbelt, Maryland 20771

8. PEFORMING ORGANIZATION REPORT NUMBER
2002-01501-1

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS (ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
CP—2002–210000

11. SUPPLEMENTARY NOTES
P C Hariharan, Systems Engineering and Security, Inc., Greenbelt, Maryland

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category: 82
Report available from the NASA Center for AeroSpace Information,
7121 Standard Drive, Hanover, MD 21076-1320. (301) 621-0390.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This volume collects together 27 papers from the Tenth Goddard Conference on Mass Storage Systems and Technologies being held in cooperation with the Nineteenth IEEE Symposium on Mass Storage Systems and Technologies.

14. SUBJECT TERMS
Magnetic tape, magnetic disk, optical data storage, mass storage, archive storage, file storage management system, hierarchical storage management software, data backup, network attached storage, archive performance, media life expectancy, archive scalability, tertiary storage, data warehousing, holographic storage.

15. NUMBER OF PAGES
333

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT
UL