SLEDs: Storage Latency Estimation Descriptors

Rodney Van Meter
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
rdv@ISI.Edu
tel: +1-310-822-1511
fax: +1-310-823-6714

Abstract

Managing the latency of storage systems is a key to creating effective very large scale information systems, such as web interfaces to satellite image databases and video-on-demand servers. Storage Latency Estimation Descriptors (SLEDs) are architecture-independent descriptions of the retrieval time of a unit of data. They describe the latency to the first byte, and the bandwidth expected. SLEDs are an important enabling technology for true end-to-end quality of service (QoS) because they can be used to predict and schedule data transfer with multimedia (guaranteed I/O rate) file systems. SLEDs provide the interface that allows database management systems (DBMS) and clients of hierarchical storage management (HSM) systems to optimize their data access patterns by choosing to read data in specific sequences or not at all. SLEDs are designed to work in intra-machine, local-area network (LAN) and wide-area network (WAN) storage systems, and to scale through twelve orders of magnitude in latency, from thousands of seconds down to nanoseconds.

1 Introduction

In this paper, we propose exposing certain aspects of storage state to allow applications to make their own determinations about storage access timing and ordering based on the current state of the data. This appears to be especially promising for use in hierarchical storage management (HSM) systems, where the cost to retrieve data can be extremely high. Our approach provides storage system state information via an abstraction known as Storage Latency Estimation Descriptors (SLEDs).

SLEDs will enable applications cooperating with storage systems to both predict their performance and improve that performance by reducing the I/O load they impose on the system. The potential reduction in I/O load comes primarily from applications electing not to perform certain I/O operations, and secondarily from reduction in thrashing due to improved cache utilization by I/O reordering.

Hints in file systems have been the topic of much recent research. Hints send information from the application to the storage system to improve prefetching and caching. SLEDs invert this process, sending information from the storage system to the application to allow applications to make intelligent, informed decisions.

A key premise of SLEDs is that CPU time is cheap, but I/O operations are expensive. During the roughly ten milliseconds required to execute a magnetic disk operation, a 100 MIPS CPU (modest by today's standards) can execute a million instructions, and current technology trends indicate that CPUs will continue to to improve faster than disk drives. Thus, spending up to a million instructions to reduce the number of I/Os by one may reduce the application running time.

In hierarchical storage management systems the access time for a given page can vary from under a microsecond for RAM-cached data to milliseconds for data on hard disk, tens of seconds for data on magneto-optical disks in an autochanger or hundreds to thousands of seconds for offline material stored in manually mounted tapes. This is a span of roughly twelve orders of magnitude. SLEDs provide methods for improving the utilization of such systems, with potential performance gains of several orders of magnitude while reducing overall system load.

SLEDs will be useful when browsing datasets kept in HSM systems – for example, large satellite image databases. Although web browsers can predict completion time as data arrives by knowing the size and transfer rate once data begins arriving, the latency to the first byte of data is usually unknown when the data is actually stored in an HSM system. Even the web server may be unable to determine how long retrieval will take. SLEDs provide the interface that allows the storage system, web server and browser to cooperate to provide the end user the information required for informed, time-efficient database browsing.

SLEDs are expected to have a broad impact on data-driven information services, including networked databases, digital libraries and video-on-demand services. Without the performance improvements and predictions SLEDs can provide, successfully deploying such services will require ad hoc solutions to these problems.

The rest of the paper is organized as follows: section 2 presents the basic technical details of SLEDs, and section 3 presents some uses. Section 4 describes the heterogeneous storage environments SLEDs support and section 5 describes implementation generations. Section 6 presents an alternative I/O paradigm built on SLEDs. Sections 7 and 8 are related and future work. Sections 9 and 10 describe ISI's qualifications to conduct such research, and a research plan should the project be funded.

2 SLEDs

SLEDs are an exposure of the file's current storage state. Using a system call, an application can retrieve information about the locations of a file's data blocks. Note that if the metadata is not memory-resident, this call may itself result in I/O being performed.

SLEDs provide:

- An open, consistent interface for storage access optimization, regardless of storage technology or location.
- Information flow about storage state from the system to applications; when combined with hints and reservations, information flows in both directions across the

system/application boundary.

- The necessary "next step" enabling technology to bring predictable performance to local, network and wide-area storage systems, supporting real-time applications and quality of service (QoS).
- A "future proof" substrate for I/O programming, because they are technology independent.
- Resource utilization improvements by allowing applications to actively participate in I/O scheduling and pruning.

The metadata is returned as an extent map with two key pieces of timing information, the expected latency and throughput. In addition, SLEDs may include an indication of the reliability level of the latency and bandwidth numbers, which should be high for local disk and low for distributed file systems, and a system state change function, which will be discussed later. While it is possible to return more complex information, such as the device ID, block address, device type, bus (or network) attachment, mount status, head position, etc., the two time estimates should be adequate for many purposes, and provide a simple, device-independent approach.

The latency includes rough estimates of the time to retrieve data from tape, when necessary. This must include some estimate of the wait time for a tape drive, robot handler, tape load and seek. This information is clearly both very dynamic and difficult to estimate accurately, and the representation of this data is an open area of research.

Key difficulties in representation include characterizing the effects of a given operation on the system state. However, this must be taken into account for the system to be complete. Given complete freedom to schedule requests, finding the optimal schedule is combinatorially prohibitive, so heuristics will be used.

SLEDs describe extents, so an entire file or dataset can consist of disjoint segments stored in various places. SLEDs are independent of whether the data is stored locally in cache, on hard disk or tape, or remotely in distributed file systems. SLEDs are "future proof" in that they describe time-to-data in an abstract fashion, not tied to the concepts of sequential or rotating media (tape or disk) or networks. Thus, code written once to work within the SLED paradigm will never become burdensome legacy code.

The current three major access effect types are true random (RAM), rotating with seek (disk), and sequential (tape). Unusual, difficult-to-characterize subsystems do exist (serpentine tapes such as DLT exhibit bizarre seek-time patterns; autochangers with rotating drums and handlers may be even more complex) and other unforeseen major types may develop (two or three dimensional positioning for holographic or micromechanical storage?). Because SLEDs are technology-independent, only a new "state change function" for a given technology must be provided in order to explore the impact of different access patterns.

3 Using SLEDs

The interaction between SLEDs and applications falls into four categories: (a) applications with flexible I/O ordering that can use SLEDs to schedule I/O in arbitrary order, (b) applications that will use SLEDs to make decisions about which I/Os to perform, "pruning" the set of I/Os actually executed in the interest of completion time or cost (for those systems that charge for I/O), (c) applications that use SLEDs just to predict performance, and (d) fixed-order algorithms with no need for prediction, which will be unable to use them effectively.

SLEDs support application-controlled access patterns. They require recoding of applications in order to realize the benefits. Applications must be willing to be flexible about the order of file requests. Many database-like programs, where the order of record execution often is inconsequential, are expected to make good use of SLEDs. This will allow better use of data currently in cache, reducing the thrashing that may otherwise occur.

The Unix `find` utility is an excellent example of a utility which will benefit from being adapted to use SLEDs by being able to "prune" its I/O request tree. `find` traverses a directory tree, looking for files with specific characteristics and performing some operation on them, such as forking off a `grep`. `find` currently provides a switch which instructs it not to follow links which will lead it into NFS-mounted file systems, because doing so can very adversely affect the performance not only of the `find` but potentially all systems connected to the same network. In HSM systems providing automatic file migration, the same can be true; the negative impact of imposing load on the HSM system is significant. If SLED-aware `find` is instructed not to access any file with a latency of more than, for example, 100 milliseconds, the find will complete more quickly and with less overall system load. As the object of `find` is often to locate one or a small set of particular files, if they have been recently used, the HSM system likely has them on low-latency storage, meaning that the SLED-unaware `find`'s approach of reading all files will be not only slow and expensive but pointless; SLED-aware find would be faster and cheaper.

For applications unable to reorder or reduce their I/O requests, SLEDs will provide only a means for estimating the performance that can be achieved, a useful feature for admissions control in real-time environments and evaluation of potential execution in multimedia file systems [25, 3].

As mentioned above, applications such as web browsers may find SLEDs' performance prediction useful, in order to let the user know how long a particular request might take, possibly giving the user the option to cancel requests for which he is unwilling to wait. Servers can also utilize performance predictions to manage movement of data among levels of a hierarchy for HSM-based video-on-demand environments [20, 10]. When combined with the Internet's ReSerVation Protocol (RSVP) at the application layer, true end-to-end quality of service guarantees can be achieved.

4 SLEDs in Different Storage Environments

In this section we briefly describe the key storage environments in which SLEDs are expected to be used: local file systems, HSM systems and distributed file systems. Although all of these provide similar programming interfaces, their performance characteristics are very different. The goal of SLEDs is to effectively manage this heterogeneity.

4.1 Local File Systems

SLEDs will find their first application in local on-disk file systems. The first level of information is knowing what is in the file system cache, and what must be fetched from disk. A second-generation SLEDs implementation will understand the page replacement algorithm so that varying request sequences can be explored. Third-generation models will incorporate physical characteristics such as head position (seek) times and zoning [27, 34, 31], while in the distant future, it could ultimately become important to model the rotational position of a drive. SLEDs can ultimately support such functionality without a change in approach.

4.2 Hierarchical Storage Management

It is in hierarchical storage management (HSM) systems that SLEDs have the potential to be most effective. In HSM systems, the access time for a given data block can vary by twelve orders of magnitude. In addition, the high-latency devices such as tape drives operate on single requests, so their transaction rate is very low. Thus, reducing the I/O load on these devices has the potential to improve the performance of not only a given application, but the entire system.

HSM systems apply heuristics to improve execution schedules, but make no attempt to involve the application in such decisions. They do not offer a portable, technology-independent interface that involves applications in the management of data movement, preferring instead to hide such operations, sacrificing system performance to a simpler, more familiar API. This transparency is a strength of HSM, but many applications will want to have the enhanced interface SLEDs provide.

SLEDs initially allow applications to prune their requests, eliminating unnecessary accesses to offline data. This can be especially useful in searching applications, where a match in online data may result in termination of the program before any requests to tertiary devices have become necessary. It is here that SLEDs offer the largest potential gains, with three orders of magnitude or more improvement possible.

In HSM systems, optimizing the access patterns for tape drives is closely tied to the ability to ability to predict their performance [15, 13, 16]. SLEDs will incorporate such information, allowing applications to explore different request sequences.

When processing datasets that exceed the size of hard disk cache available, successive passes across the data are likely to find the cache in different states. By choosing to process readily-available data first, the total number of requests from tertiary storage (and, roughly, execution time) required to scan the entire data set can be reduced by $1/N$, where N is the fraction of the data set that will fit in cache.

As in local file systems, the first class of information provided is an indication of the current resident level of desired information. With just this simple information, the pruning suggested above is possible. The second class of information provided will incorporate some knowledge of cache replacement as data from tape is brought into disk caches. The third generation of SLED will incorporate mechanical knowledge of the complex devices, such as autochangers and tape drives, involved. This will allow direct estimation of the completion time for specific request sequences.

The High Performance Storage System [32] provides an interface for hints, but it is currently unimplemented; SLEDs could be incorporated as well. Systems based on the Open Storage Systems Interconnect model [17] provide *virtual stores*, and SLEDs can be used to model their characteristics.

Menon and Treiber have asserted that programmers will be unwilling to optimize their code to improve access patterns and performance in HSM systems, likening the difference between tape and disk to the difference between memory and disk [19]. However, the gap between memory and tape is larger, and therefore, more likely to produce significant performance improvements for modest programming effort.

4.3 Distributed File Systems

Applying SLEDs to distributed file systems is an interesting problem, due to the cooperation required between client and server. Ideally, both the client and the server will support SLEDs, in which case the client can request the SLED for a particular data segment, and adjust for the network performance. When the client supports SLEDs but the server does not, the SLED for data not locally cached can indicate no better than “remote”, or utilize past history to generate an estimate.

SLEDs for distributed file systems will require real-time network protocols and techniques such as RSVP to create a complete end-to-end quality of service. This will create, effectively, NFS with the guaranteed I/O rates now provided by multimedia file systems [25, 3, 29].

In a distributed environment, clients have less knowledge and control of activity at the server, and unpredictable network traffic can interfere. Thus, the SLEDs in this case will indicate higher potential variance in performance.

5 Evolving SLEDs

SLEDs do not have to be perfect in order to be useful; over time, the accuracy of the SLEDs representations for specific devices and systems will gradually improve, as will the heuristics for choosing schedules. Thus, we expect the overall SLEDs concept to pass through several generations.

The details of the incorporation of change in overall system state due to requests is one of the most open areas in SLEDs. The earliest SLEDs will simply represent levels of the hierarchy, with no indication of changing state. Second-generation SLEDs will represent changes in the system state based on speculative execution of specific sequences, primarily by understanding the cache and page replacement algorithms. Third-generation SLEDs will incorporate physical knowledge of device performance, including tape motion, robotic autochanger motion and disk head seeks.

This system state change representation may have to take the form of executable code associated with a specific SLED, and make modifications to a memory-resident state diagram. The exact form for this is yet to be determined. While this may sound computationally expensive, modest CPUs, by today’s standards, can already execute billions of operations in the time required to load a tape and seek, and this ratio will continue to increase.

```
frio_reset_file_record_counter(fd);
while (frio_record_remaining(fd)) {
  /* might come in in any order */
  frio_record_read(fd,buffer);
  process_record(buffer);
}
```

Figure 1: Code Fragment Using Free-Range I/O

6 Free-Range I/O

In this section we propose a new data storage paradigm, more akin to a persistent object store than the traditional Unix-style linear byte stream. We have dubbed this approach *free-range I/O*.

If a file is defined as an unordered collection of records, the system may be able to take better advantage of the current storage state in order to fulfill the application's requests. As with a database, the application may request any aribtrary record which it has not seen, giving the storage system the freedom to choose the easiest-to-access record to return.

Figure 1 shows one possible example of how free-range I/O might be used. SLEDs can provide the substrate on which it would be possible to build such an API. Such an API may simplify the process of coding applications to use SLEDs, as described in section 3. Free-range I/O may be implemented as a library over top of flat files, in typical Unix fashion, or by directly modifying the underlying storage structures. SLEDs provide the infrastructure to optimize record access ordering in free-range I/O.

7 Related Work

Multimedia file systems and video on demand storage servers [23, 6, 11, 12, 22, 3, 25] generally provide a mechanism for the application to communicate its needs to the storage system, but provide no feedback in the opposite direction, limiting the level of cooperation.

RSVP is the ReSerVation Protocol under development for supporting Integrated Services on the Internet [5]. Smooth integration of SLEDs with RSVP is paramount, but there are some important differences. RSVP is designed to support policing mechanisms and packet-based network jitter control, which are complementary to the problems SLEDs solves but perhaps not necessary in SLEDs themselves. RSVP has no "state change function" associated with accessing certain data, and deals with streams of data rather than the specific data objects of SLEDs.

Joe Touch has suggested a Resource Descriptor Interface [30], which proposes a model for representing all system resources as communications links. SLEDs are similar, but are more closely tied to the concept of data objects with storage and system state.

Hints are used by file systems to order disk accesses and make caching decisions. These hints can be provided explicitly [24, 1, 8] or determined by the system based on information such as file name or access history [14]. Hints have also been implemented across a network [26]. However, hints are typically one-way information; there is no way for the application to base its I/O patterns on the current state of the file system cache and storage devices.

Systems as far back as Tops-10 [9] have supported user-definable pagers, as does Mach. Application-controlled file caching [7] is most like SLEDs, but is tied to the concept of on-disk file systems, with no support for multiple storage hierarchy levels, and no ability to predict performance.

Tops-20, NAStore and Cray's Data Migration Facility [33] support a single-bit addition to the file system's inode to indicate that files are in archival storage rather than on disk. This aproach can be viewed as very simple SLEDs. However, many modern HSM systems support partial file caching on magnetic disk and multiple levels in the storage hierarchy, including optical disk, robotic tape and offline tape, with orders of magnitude difference in latency and bandwidth. The difference among these is important and not represented by a simple per-file bit.

SLEDs provide a way to characterize the performance of disparate devices, so that combined disk/tape operations can be written in a more device-independent fashion, providing a better growth path as device technologies continue to change. In some database systems the data is laid out explicitly on both disk and tape, and access methods must explicitly refer to one or the other [21]. Mainframe systems have explicitly made the distinction between data on tape and data on disk for years, while actively using tape to manipulate (as opposed to backup and restore) data. Sorting algorithms based on tape's sequential access method have also been used for many years [18]. SLEDs may provide the infrastructure for understanding the delay in retrieval necessary to support economic query optimization models in database systems [28].

HP's attribute-managed storage [4] incorporates performance descriptions of the storage devices to be integrated into a storage system. Their goal is to meet system-wide performance and data integrity goals through monitoring and feedback, providing quality of service by adjusting the system configuration over time. Synergistic relationships between SLEDs and attribute-managed storage are worth pursuing.

8 Future Work

Much work in defining the role of SLEDs in storage systems remains. Many details of the functionality need to be established, and SLEDs needs to be implemented and studied to make the ultimate determination of whether or not the performance improvements they appear to offer can be truly realized with modest programming effort.

The primary unresolved technical problem is representing the change in the state of the system caused by requests, as described above. However, even simple heuristic solutions offer the promise of improved performance.

In addition to managing one's own sequence of requests, the overall storage system state will of course be affected by other activity in the system. How can this be taken into account? Do SLEDs need to timeout, or provide an interface for requesting guaranteed performance through a global reservation mechanism of some sort?

The SLED model must also support replication and striping of data effectively. The data structures for this appear straightforward, but insuring that applications are aware of it and can take appropriate advantage is more complex.

An interesting question is whether or not SLEDs can be applied to memory systems, as well, where latencies can vary by two orders of magnitude or more from cache to main memory to remote memory, in distributed shared memory systems or non-uniform memory access (NUMA) architectures. SLEDs with free-range I/O could ultimately prove a useful enabling technology for data flow-like operating systems and languages.

SLEDs as presented here represent primarily a means for managing read access to data; the paradigm may be extended to writes as well by providing a reservation/hinting type of interface. The system must take into account that many devices have different read and write rates.

Since SLEDs offer a broad paradigm shift, much work in realizing them remains before a final determination of their worth can be made.

9 Implementation and Deployment Plans

In order to successfully deploy SLEDs, several technical problems must be solved, and a sufficient base of applications and knowledgable applications programmers must be built. The key problem is representing changes in system state, both those scheduled by the SLEDs-aware application and those initiated by other, unrelated clients of the storage system. In addition, for more effective use in networked environments, wider use of technologies such as RSVP is required.

SLEDs are expected to be deployed first as a performance prediction tool for use in multimedia and web servers. As experience is gained and the application base builds, SLEDs will gradually affect many data-intensive environments, such as databases.

SLEDs, although they represent a significant shift in I/O programming, can be deployed incrementally. The sophistication of applications, kernel services and device SLEDs are expected to go through several generations, all capable of coexisting.

10 Conclusion

In this paper, we have proposed Storage Latency Estimation Descriptors (SLEDs) as a means to more directly involve applications in the management of data movement in heterogeneous storage environments. SLEDs represent the latency and bandwidth to any given segment of storage, representing latency across twelve orders of magnitude. When utilized with hierarchical storage management systems, SLEDs have the potential to improve performance of applications by orders of magnitude by allowing them to intelligently control their own access patterns. SLEDs exploit the increasing imbalance between CPU and I/O device speeds by utilizing the former to improve utilization of the latter, in a device- and technology-independent fashion, so that SLEDs will remain a viable paradigm as storage technology evolves in unforeseen ways.

Acknowledgements

This research was sponsored by the Defense Advanced Research Projects Agency under Contract No. DABT63-93-C-0062. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policies, either expressed or implied, of ARPA, the U.S. Government, or any person or agency connected with them.

References

[1] High performance storage system design specification for client API delivery 2.0 draft 3.0, June 1994.

[2] ACM. *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, June 1996.

[3] D. P. Anderson, Y. Osawa, and R. Govindan. A file system for continuous media. *Trans. on Computing Systems*, 10(4):311–337, Nov. 1992.

[4] E. Borowsky, R. Golding, A. Merchant, E. Shriver, M. Spasojevic, and J. Wilkes. Eliminating storage headaches through self-management. In *Proc. Second USENIX Symp. on Operating Systems Design and Implementation*. USENIX, Oct. 1996. work-in-progress abstract.

[5] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – version 1 functional specification. Internet draft, Nov. 1996.

[6] D. W. Brubeck and L. A. Rowe. Hierarchical storage management in a distributed VOD system. *IEEE Multimedia*, 3(3):37–47, 1996.

[7] P. Cao, E. E. Felten, A. R. Karlin, and K. Li. Implemention and performance of integrated application-controlled file caching, prefetching, and disk scheduling. *ACM Trans. Comput. Syst.*, 14(4):311–343, Nov. 1996.

[8] P. Corbett, D. Feitelson, S. Fineberg, Y. Hsu, B. Nitzberg, J.-P. Prost, M. Snir, B. Traversat, and P. Wong. Overview of the MPI-IO parallel I/O interface. In R. Jain, J. Werth, and J. C. Browne, editors, *Input/Output in Parallel and Distributed Computer Systems*, chapter 5, pages 127–146. Kluwer Academic Publishers, 1996.

[9] Digital Equipment Corporation. *decsystem10 Monitor Calls*, June 1976.

[10] C. Federighi and L. A. Rowe. A distributed hierarchical storage manager for a video-on-demand system. In *Proc. Storage and Retrieval for Image and Video Databases II, IS&T/SPIE Symp. on Elec. Imaging Sci. & Tech.*, Feb. 1994.

[11] D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L. A. Rowe. Multimedia storage servers: A tutorial. *IEEE Computer*, 28(5):40–49, May 1995.

[12] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. Pipelining mechanism to minimize the latency time in hierarchical multimedia storage servers. *Computer Communications*, 18(3):170–184, Mar. 1995.

[13] J. J. Gniewek. Evolving requirements for magnetic tape data storage systems. In B. Kobler, editor, *Proc. Fifth NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 477–491, Sept. 1996.

[14] J. Griffioen and R. Appleton. Performance measurements of automatic prefetching. In *ProceedingsInternational Conference on Parallel and Distributed Computing Systems*, Sept. 1995.

[15] B. Hillyer and A. Silberschatz. On the modeling and performance characteristics of a serpentine tape drive. In *Proc. ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 170–179. ACM, May 1996.

[16] B. K. Hillyer and A. Silberschatz. Random I/O scheduling in online tertiary storage systems. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* [2], pages 195–204.

[17] IEEE P1244. *Reference Model for Open Storage Systems Interconnection – Mass Storage System Reference Model Version 5*, Sept. 1994.

[18] D. E. Knuth. *The Art of Computer Programming, volume 3 / Sorting and Searching*. Addison-Wesley, 1973.

[19] J. Menon and K. Treiber. Daisy: Virtual-disk hierarchical storage manager. *Performance Evaluation Review*, 25(3):37–44, Dec. 1997.

[20] T. Mori, K. Nishimura, Y. Ishibashi, and H. Nakano. Video-on-demand system architecture using an optical mass storage system. In *Proc. Joint International Symposium on Optical Memory and Optical Data Storage*, pages 41–42, July 1993.

[21] J. Myllymaki and M. Livny. Disk-tape joins: Synchronizing disk and tape access. In *Proc. ACM SIGMETRICS Conference*, May 1995. Expanded version retrieved via ftp.

[22] W. O'Connell et al. A teradata content-based multimedia object manager for massively parallel architectures. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* [2], pages 68–78.

[23] B. Ozden, R. Rastogi, and A. Silberschatz. Architecture issues in multimedia storage systems. *ACM Performance Evaluation Review*, 25(2):3–12, Sept. 1997.

[24] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proc. 15th ACM Symposium on Operating Systems Principles*, pages 79–95. ACM, Dec. 1995.

[25] P. V. Rangan and H. M. Vin. Designing file systems for digital video and audio. In *Proc. Thirteenth ACM Symposium on Operating Systems Principles*, pages 81–94, Oct. 1991.

[26] D. Rochberg and G. Gibson. Prefetching over a network: Early experience with CTIP. *Performance Evaluation Review*, 25(3):29–36, dec 1997.

[27] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *Computer*, 27(3):17–28, Mar. 1994.

[28] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in Mariposa. In *ProceedingsParallel and Distributed Information Systems*, Austin, Texas, Sept. 1994.

[29] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. In *Proc. 1996 USENIX Technical Conference*, pages 1–14. USENIX, Jan. 1996.

[30] J. Touch. A view of communication middleware. presentation at Sigcomm '95 invited workshop on middleware, Aug. 1995.

[31] R. Van Meter. Observing the effects of multi-zone disks. In *Proc. USENIX '97 Technical Conference*, pages 19–30. USENIX, Jan. 1997.

[32] R. W. Watson and R. A. Coyne. The parallel I/O architecture of the high-performance storage system (HPSS). In *Proc. Fourteenth IEEE Symposium on Mass Storage Systems*, pages 27–44. IEEE, Sept. 1995.

[33] T. S. Woodrow. Hierarchical storage management system evaluation. In B. Kobler and P. Hariharan, editors, *Proc. Third NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 187–216, Oct. 1993.

[34] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling for modern disk drives and non-random workloads. Technical Report CSE-TR-194-94, University of Michigan, Mar. 1994.