ACM Computing Surveys, **28** (4), December 1996, also available as http://www.cs.duke.edu/~jsv/report.ps.

Report of the Working Group on Storage I/O Issues in Large-Scale Computing

ACM Workshop on Strategic Directions in Computing Research

Edited by Garth A. Gibson (CMU), Jeffrey Scott Vitter (Duke), and John Wilkes (HP Labs)

With participation from Alok Choudhary (Northwestern), Peter Corbett (IBM), Thomas H. Cormen (Dartmouth), Carla Schlatter Ellis (Duke), Michael T. Goodrich (Johns Hopkins), Peter Highnam (Schlumberger), David Kotz (Dartmouth), Kai Li (Princeton), Richard R. Muntz (UCLA), Joseph Pasquale (UCSD), M. Satyanarayanan (CMU), and Darren Erik Vengroff (Delaware).

1 Introduction

In this report we discuss the strategic directions and challenges in the management and use of *storage systems*—those components of computer systems responsible for the storage and retrieval of data. Typical large-scale storage systems include the following components of secondary and tertiary storage:

- rigid or hard magnetic disks;
- parallel disk arrays;
- optical disks, which come in several variants: read-only (such as CD-ROM and Digital Video Disk (DVD)), write-once (WORM), and rewritable (magneto-optical or phase change);
- magnetic tape, such as 4mm (DAT), 8mm, and Digital Linear Tape (DLT);
- autochangers, which combine storage devices with racks of tape cartridges or optical disk platters that can be moved to and from the drives by robotics.

Complete storage systems also include software to manage and orchestrate the storage components. Such software includes management and configuration utilities for storage devices, logical volume managers that tie multiple physical devices together as one, and file systems to arrange layout of data on storage devices.

Storage systems represent a vital and growing market. Their primary components are magnetic and optical disk drives, magnetic tapes, and large-capacity robotic assemblies of drives and cartridges. Storage hardware sales in 1995 topped $40 billion, including more than 60,000 terabytes of hard disk storage. In recent years, the amount of storage sold has been almost doubling each year; in the near future it is expected to sustain an annual growth of about 60 percent. This enormous growth rate has been accompanied by a 50 percent per year decrease in the cost per byte of storage. These growth trends exceed those of the personal computer market.

The focus of this report is on secondary storage systems, and primarily magnetic disk drives, since these are the dominant storage devices commercially. Many of the issues discussed here arise also in tertiary storage systems that use tape drives and autochangers, where there is an even greater disparity between main memory performance and tertiary device access times. For more information on this area, we refer the reader to the proceedings of the IEEE Symposia on Mass Storage Systems and work elsewhere that touches on the integration of tertiary storage systems into an overall storage hierarchy (e.g., [34, 37, 47, 66]).

In the next section we discuss the issues and problems in large-scale storage. An important set of broad strategic goals are identified in Section 3, and relevant research directions are outlined in Section 4. We make some concluding remarks in Section 5.

2 Storage Challenges

To what use is the growing amount of storage being put? Storage capacity is being consumed in several ways:

- *Cached data.* Local copies of data and applications provide privacy, speed and availability. For example, almost all personal computers contain a hard disk that holds a local copy of the operating system and many, if not all, of the applications used.
- *Historical data.* In the past it was often the case that only the most recent, active data could be stored. Nowadays it is frequently cost-effective to store historical data (including audit trails) on-line, so as to allow operations such as data mining and trend analysis.
- *Multimedia.* The recent increase in the generation, capture, and storing of digital multimedia (video, images, and sound) is consuming a disproportionate amount of storage capacity because of the relatively large sizes of these data types. For example, it is estimated that the fraction of the total available storage allocated to digital multimedia will grow from 10 percent in 1995 to 50 percent by 2005.
- *Scientific data.* The amount of large-scale scientific (especially geometric) data from geographic information systems (GIS), seismic databases, and satellite imagery is also growing rapidly. NASA's EOS project will collect petabytes (thousands of terabytes) of spatial data and imagery.

As the ubiquity of data storage has grown, so too have user expectations for it. It is no longer enough simply to store data. High performance access to data must be provided, regardless of the location of the data and users or of the nature of faults that may be interfering with access.

Unprecedented levels of reliability are needed for modern storage systems, reflecting the growing importance of information as a commercial and national asset. Businesses in particular are coming to rely on their stored information to an ever-increasing degree. As recent floods and other natural disasters have demonstrated, companies that are suddenly unable to access their information base can have considerable difficulty surviving.

Not only is stored data becoming more crucial, but the applications that use the data are becoming both more heterogeneous and more demanding in their needs. For example, applications often expect to be able to make greater demands for higher bandwidth, increased capacity, and lower latency than was the case in even the recent past. New algorithms and architectures put different demands on the I/O subsystem.

Although the range of issues that storage system designers face is wide and varied, as we hope to show in this report, it is possible to isolate four primary problems—performance, persistence and reliability, scale, and ease of use—that arise in several settings.

Performance. A typical disk drive is a factor of 10^5–10^6 times slower in performing a random access than is the main memory of a computer system. Much of the difficulty and complexity in storage system design results from attempts to find ways of masking this enormous discrepancy in access time. Although disk performance is improving, it is doing so unevenly. Bandwidths have recently been increasing at about 40 percent per year after a long period of stasis, but the positioning time (the time needed to get the disk head and media to the right place) has seen much smaller improvements because it is largely caused by mechanical motions that are subject to physical bounds on inertia, acceleration, and arm stiffness. The result is that the performance discrepancy between disk and main memory access times, typically called the *access gap*, will likely be with us for quite a long time to come.

Amdahl's law tells us that I/O performance must track compute performance if I/O is not to become the bottleneck in high performance computation. There are really only four techniques available for solving the performance problems:

- increasing storage device parallelism, so as to increase bandwidth between storage I/O devices and main memory;
- more effective caching and, more generally, reorganizing data within the storage system to exploit locality, thereby reducing the cost of accessing data;
- overlapping I/O with computation (for example, by prefetching data before it is needed) in order to reduce the time that an application spends waiting for data to be input from or output to a slow storage device; and
- more effective scheduling and, more generally, reducing or rearranging the accesses made to data, possibly by changing the applications themselves.

The complexity of hiding the access gap is exacerbated by the need to provide some applications with guarantees on latency or continuous bandwidth. Examples of such needs arise in continuous media data types such as digital video. The bandwidth requirement allows a video stream to be kept running without having to slow down the replay; the latency requirement bounds the amount of buffer space needed to avoid dropping frames.

Persistence and Reliability. Storage systems are expected to be *persistent*, by which we mean that the data they store survives across system restarts. Storage systems are also expected to be *reliable*—that is, not lose data—across a multitude of failures, such as software errors, processor, network and power supply outages, storage device failures, operator error, and complete site outages. The data must be made *available* to users and applications whenever it is needed, despite these potential failure modes. The primary technique here is the use of full or partial redundancy for data, the processors that access it, and the connections among components.

A related property, usually the shared responsibility of storage and application systems, is the *integrity* of data, that is, the degree to which the contents of data (the bits and their meaning) are not damaged by having been stored and retrieved. For example, it is usually an application's responsibility to eliminate the dangling pointers that can be created when memory structures are stored and accessed in distinct pieces [79].

Scale. Large systems bring new problems and exacerbate old ones. The multitude of devices needed for large storage systems are hard to configure, monitor, and manage. Examples of very large scale include multi-terabyte global information systems, geographic information systems (GIS), grand challenge datasets, and digital libraries. Capacity is not the only problem, however. There are enormous bandwidth needs for applications running on massively parallel high-end supercomputers, large-scale workstation clusters, multimedia servers, and data-mining systems. It is very difficult to coordinate the storage, network, and computation resources needed to meet the bandwidth and latency needs of hundreds or thousands of concurrent users in such large-scale systems.

An increasingly important issue of scale for storage is locating, or naming, desired information. Large capacity in storage systems usually implies a large number of named data collections. Collections of storage systems often share a common namespace, and in the case of the world wide web, data managed by diverse machines, applications, and access technologies all share the same fast-growing namespace. This problem is compounded by on-line tertiary archives; not only must a name be found for the data sought, but the properties of its storage, such as its physical media and logical representation, need to be retrieved and the proper conversions must be identified. Namespace support, such as access-control security and type, keyword/tag or full-text search, is

becoming a generic need of storage systems, not something that should be provided separately in each application domain.

Ease of Use. The user and programming abstractions provided for storage systems must mask the complexity of hiding the access gap and managing scale. At the same time they need to provide opportunities for applications or users to disclose additional information to the storage system when doing so can significantly improve performance. Many applications handle massive data sets that are many times larger than main memory. Practical—let alone optimal—execution of such algorithms often involves careful choreography of multiple concurrent data movements. To increase the number of such applications, this data movement needs to be specified in a high-level way so that the applications become easier to write.

The above four issues cannot be addressed independently; they often impact one another. For example, making storage I/O systems easier to use via high-level abstractions can hurt performance. Much of the complexity in storage systems stems from attempting to meet the goals of persistence and reliability at the same time as the performance goals. For example, writing updates lazily to disk can reduce I/O traffic and allow better device scheduling through request reordering, yet it exposes data to loss from power or component failures, and it can allow partial, inconsistent updates to occur.

The field of storage systems spans a wide range of disciplines, providing a common ground on which these different disciplines meet, as illustrated in Figure 1. We distinguish the field of storage systems from the field of information systems and from specific application domains by their degree of structure: Storage systems operate on collections of largely uninterpreted data bytes, whereas information systems and application domains generally understand and interpret the information that they maintain. As systems evolve, storage systems are beginning to deal with structured data, and there is a continuum of degree of content understanding from the I/O system to the application system. Work in storage systems therefore benefits from and significantly enriches the theory and practice of a multitude of fields. Some of the relevant fields, together with illustrative examples of the areas and techniques that overlap with storage I/O, are the following:

- databases and information systems: geographic information systems (GIS), digital libraries, B-trees, object stores, transactions, journaling, query optimization, data mining;
- fault-tolerance: mirroring, RAID, multi-pathing;
- computer architecture: buses/channels, I/O processors;
- real-time/multimedia systems: video servers, guaranteed bandwidth;
- parallel computing: out-of-core (external memory) applications, parallel file systems, predictable access patterns, workstation clusters;
- algorithm theory: out-of-core (external memory) algorithms, parallel disk models, sorting and searching, computation and access (co)scheduling;
- compiler and runtime systems: programming environments, data tiling and blocking;
- operating and file systems: clustering, multi-level indexing, buffer caches, readahead algorithms, virtual memory paging, storage hierarchy access and control algorithms; and
- distributed systems: parallel, concurrent, mobile, non-local file system designs.

Detailed discussions of the above fields can be found in the reports of the respective working groups of this workshop.

Figure 1: Storage I/O systems research exchanges ideas with, uses results from, and enriches many other computing domains and techniques. This figure displays a few such areas of overlap.

3 Strategic Research Directions

In this section we identify, in no particular order, some strategic long-term goals for the storage systems of the future, which will be charged with meeting the challenges described in Section 2:

- unify I/O techniques;
- exploit storage metadata;
- self-manage storage;
- accommodate multiple service classes;
- rapidly develop new technologies and techniques.

3.1 Unifying I/O techniques

The storage access and management techniques developed in diverse computing domains should be codified and unified into a cogent body that can be retargeted as needed. Although related to a standards effort that has been underway for the last decade [34], this strategic direction is not necessarily about standards; the goal of unification of I/O techniques is to enable new I/O intensive applications to build on the successful techniques developed specifically for other application domains. Emerging data access applications dealing with the world wide web or mobile computing should be within the scope of these techniques.

3.2 Exploiting storage metadata

Structural information (or metadata) describing stored data should be acquired and broadly exploited. Existing type information should be recognized and algorithms developed to use this information to meet type-specific requirements. New metadata information types, such as for quality of service for storage, and algorithms to exploit them, may be needed to meet emerging requirements.

3.3 Self-management

Storage management tasks such as (re)configuration, backup, and capacity-balancing or load-balancing are often performed by human administrators. These tasks should be extracted from

application systems and aggregated into the storage system. Wherever possible, management techniques should statically and dynamically adapt to the available resources and mix of application workloads.

3.4 Accommodating multiple service classes

The emerging requirements for high availability and for guarantees on access latency and bandwidth are distinct from one another and from the existing requirements for maximum sustained bandwidth and minimum access latency. Storage systems must accommodate these multiple classes of service. Parallels with techniques in the networking community should be sought and exploited.

3.5 Rapid adoption of new technologies

New storage device technologies such as Digital Video Discs (DVD), holographic crystal arrays, and micromechanical magnetic arrays, along with new software and management techniques needed for their use, should be sought out and exploited to provide users a widening range of storage quality trade-offs and to reduce the access gap between main memory and existing storage technology performance.

4 Technical Research Directions and Approaches

In this section we outline several promising technical approaches to address the strategic goals of Section 3. These approaches include

- richer semantics and information hiding for various storage interfaces, via
  - virtual device interfaces,
  - new programmer abstractions;
- pushing or pulling customization information through these interfaces by
  - exploiting data types and access patterns,
  - quality of service negotiation;
- sophisticated prefetching, caching, and scheduling;
- infrastructure changes to allow efficient implementations of the above approaches, such as
  - detailed performance and behavior models,
  - exploiting idle processing power,
  - application influence over operating system policies.

4.1 Virtual device interface

A *virtual device interface* is a programmatic interface that allows the hiding of one or more features of the underlying storage system's implementation. It can be used for many purposes. For example, in the Small Computer System Interface (SCSI), the physical characteristics of a hard disk such as cylinder, track, and sector are hidden behind the virtual device interface provided by a linear block address space. This indirection enables devices to pack more data onto outer tracks than fits on inner tracks and to distribute spare sectors so that a replacement sector can be near a faulty sector, without having to modify the applications that use the device (in this case, operating systems software).

Virtual device interfaces are used by disk arrays (commonly called Redundant Arrays of Independent Disks, or RAID [9]) that provide full or partial redundancy for stored data to increase its availability. By doing so, they are able to hide the details of the location of redundant information and its relationship with non-redundant data (for example, which blocks are in a mirrored pair or parity set) [42]. Furthermore, disk failures can be transparently identified; the data from the broken disks can be automatically reconstructed onto spare disks, and future accesses remapped to the new devices [32].

Storage devices are typically burdened by long positioning times, and a virtual device can be used to dynamically remap the physical location associated with a logical block, thus reducing the current access latency [19, 23, 57, 58, 73]. Additionally, most modern disk drives perform dynamic request reordering, in some cases taking advantage of low-level information available only inside the storage device to optimize the request sequencing [35, 63].

Since there is no single redundant disk array organization that is optimal for all accesses, the choice of redundancy scheme used to protect data can be dynamically selected so as to balance competing demands for low cost, high throughput, and high availability [48, 61, 76].

Using a virtual device model, self-managing storage systems may migrate into the storage system much of the low-level device-management functions that are traditionally exported into client operating systems. This might allow, for example, a storage subsystem to accept new devices or workload requirements and transparently (re)configure itself to utilize the resources of the new device or meet the workload requirements [43, 30].

As these examples demonstrate, virtual device interfaces have already been exploited effectively in several ways. The flexibility of virtual device interfaces ensures that they will play a major role in emerging techniques for self-managing storage systems.

4.2 New programmer abstractions

There are a wide range of programmer abstractions for storage. The simplest abstraction, virtual memory, offers the programmer a simple and elegant model that largely hides the notion of storage, but can cause performance to suffer.

Most programming languages provide a single-dimensional character or block access interface, called a read-write interface, that decouples data access from computation. While this interface encourages programmers to organize data in files that will be accessed sequentially (an efficient access pattern for most individual storage devices), it can also lead to access patterns and load/compute/store cycles that cause excessive I/O, in some cases bring computation to a near halt.

Techniques like caching are extremely effective when they can exploit common locality patterns in data accesses to reduce access time. Nonetheless, there are I/O-intensive applications where explicitly anticipating storage parallelism and coordinating the scheduling of memory and data accesses are essential for overall performance. Much effort will be needed in designing I/O-efficient algorithms that use fundamentally new approaches than their internal-memory counterparts. Linear algebra methodologies are useful in designing I/O-efficient algorithms that manipulate data obliviously in regular patterns [16, 44], and techniques based on simulating parallel algorithms and rearranging data by sorting have been successful for adaptive processing and irregularly structured data [11]. Higher-level abstractions, perhaps implemented by compilers and runtime systems, will be needed to provide storage access semantics and support for these new I/O-efficient algorithms and the applications that use them.

There are currently three classes of such high-level abstractions being pursued: extensions to the *access-oriented* existing interfaces; language-embedded operations on specific types, mainly *array-oriented*; and a building block approach binding programmer-provided operations to access-oriented interfaces, called *framework-oriented* interfaces. Access-oriented extensions to the read-

write interface typically include data type specifications and collective specification of multiple transfers, sometimes involving the memories of multiple processing nodes. These interfaces, possibly integrated into parallel programming toolkits, preserve the programmer abstraction of explicitly requesting data transfer [12, 13, 14, 62, 65].

Array-oriented (or type-oriented) interfaces [15, 69] define compiler-recognized data types (typically arrays) and operations on these datatypes. Out-of-core computation is directly specified and no explicit I/O transfers are managed by programmers. Array-oriented systems are effective for scientific computations that make regular strides through arrays of data.

Framework-oriented interfaces [70] extend access-oriented interfaces by providing a set of efficient high-level data access methods that manage the redistribution of storage data, requiring programmers only to specify the operations to be done on the data as it is redistributed. Framework-oriented interfaces can be effective for irregular memory access patterns.

By a similar extension of the storage abstraction, object-oriented storage (or object stores) enhance the integrity of persistent data (that is, its resistance to loss of semantic meaning) through type-safe, object-oriented computational models [8, 33, 45, 52, 64, 79].

The efficiency of array-oriented, framework-oriented, and object-oriented storage systems depends critically upon the appropriate mapping of the higher-level semantics to the functionality of the lower levels of the I/O system. This means that I/O system architecture must influence the structure of these abstractions, and perhaps vice-versa.

4.3 Exploiting data types and access patterns

A growing phenomenon is the "publication" of stored data, in which format and interpretation conventions of data are made public, and applications can recognize and interpret the data they find. Examples are diverse: MPEG is a video format, GIF is an image format, and HDF is a hierarchical data format for scientific data sets. If storage systems could recognize and exploit this structure information where it is useful, they could provide more effective application support.

Similarly, the access patterns of I/O-intensive applications, within and across files, are increasingly predictable [1, 17, 40]. For example, matrix subroutines often access data in regular patterns called strides. Full text search sequentially processes each file in a set. Incremental compilation and linking generally operate on the same set of files and library modules during each invocation. An important approach to increased application customization is recognizing and exploiting these predictable access patterns [20].

Data types and access patterns are closely related in that we can use a data type to anticipate the data's access pattern. Wherever possible, transparent extraction of storage attributes or access patterns of specific applications is desirable [31, 68]. Transparency reduces the impetus of programmers to specialize for particular storage capabilities, avoids the costs of retrofitting old codes and retraining programmers, and avoids establishing new interdependencies between applications and storage that may become inappropriate or inefficient as technology changes. In some situations, however, the information that can be extracted transparently comes too late, too inaccurately, or at more cost than value. Alternatives to the transparent learning of types and access patterns include analysis tools in compilers and explicit hints by applications. For example, a compiler for code that processes a memory-mapped matrix structure has been shown to anticipate data needed by future iterations as it processes current iterations [50]. Applications that plan accesses before executing them, such as search, visualization, and query processing, are able to deliver the access plan to their storage system; this approach lends itself to more detailed and accurate information with less delay.

4.4 Quality of service negotiation

The quality of service abstraction developed in the networking community is appealing because it addresses issues similar to ours, such as latency, bandwidth, and reliability [25]. If we group the requirements of storage access and management into distinct classes of service, the most relevant parameters of each class can be identified and passed to storage as access is initiated or space is allocated. A few distinct classes of service are already clear: best-effort high-throughput for accessing large amounts of data, best-effort low-latency for accessing small amounts of data, highly reliable verified-update of valuable data, and guaranteed bandwidth for continuous media. In many cases, a great deal of additional information is available to describe the goals or behaviors of clients. Such needs can be captured by associating attributes with the storage or its accesses, and used to drive selection of storage device, placement, and dynamic policies such as caching [26, 30, 77]. Research problems here include finding the correct way to specify client needs and storage device behaviors and how best to map one to the other.

Service quality specification may be done interactively through an explicit negotiation step, enabling the requesting application to modify its request based upon the capabilities offered by the storage system. For example, a full-motion video viewer application may request a certain guaranteed bandwidth for a stream. The storage system may refuse this, but instead counter-offer a guarantee if the application is willing to use a more appropriate request size. Similarly, a best-effort access may inquire of the storage system the access size and alignment that leads to highest throughput [5, 53]. The negotiation interface can be extended to handle dynamic changes in the application requirements or the available resources that may nullify a pre-negotiated service quality. Prefetching and caching policies on mobile computers with variable network connectivity are expected to need such cooperative dynamic adaptation [51]. With a collaborative partnership between applications and the storage system, individual applications can determine how best to adapt, but the storage system can still monitor resources and enforce allocation decisions.

4.5 Sophisticated prefetching, caching, and scheduling

One of the primary uses of the expanded metadata provided by the above type, access-pattern, and service-class acquisition techniques will be to customize storage-access scheduling and cache-replacement decisions. Prefetching and write-behind can hide access latency by overlapping accesses with computation or other accesses, increasing exploitation of parallel storage resources, and migrating storage accesses to less busy periods [39].

Caching policies can lessen the cost of accessing storage and can help balance the load on storage devices. Scheduling can improve device throughput by reordering accesses for lowered positioning times, balance the use of resources over multiple requesting applications, and reduce processing overheads by coalescing distinct accesses into a single larger access [38, 55]. For example, algorithms have been developed for efficient use of device parallelism and cache resources, given sufficient access pattern information or hints [7, 36, 54]. Useful extensions would be to support multiple active streams with different information sources (type, access pattern, service parameter), different accuracies of advance knowledge, and different service requirements.

Storage systems for continuous media, and the associated bandwidth and latency guarantees, make heavy use of scheduling to maximize storage efficiency while providing as much predictability as possible [4, 56].

Network-resource scheduling theory may be helpful for advances in storage management, as there are several parallels: In networking terms, advance knowledge of an access pattern is a transfer specification, each storage device and requesting application is a source or destination, and the cache and interconnect resources of the storage subsystem must be scheduled to accommodate concurrent classes of service.

4.6 Detailed performance and behavior models

To one degree or another, all of the strategic goals for storage systems benefit from better models of application behavior and subsystem capabilities. For example, self-management of the allocation of guaranteed bandwidth objects onto disk devices requires detailed models of the expected demand for each object, the expected behavior of each device faced with these demands, and the set of behaviors that meet the given guarantees. Models of all levels of abstraction will be needed: workload distributions, extensive application traces, device and subsystem simulations, mathematical models of service class interactions, asymptotic behavior, and average-case and worst-case bounds [2, 10, 41, 59, 60, 71, 78]. Rapid progress in the modeling area is important to some strategic goals. For example, the utility of specific data-type or access-pattern information depends upon the existence of models exploiting this information. New storage subsystem designs such as HP AutoRAID or StorageTek Iceberg have much more complex behavior than prior subsystems such as single hard disks or RAID level 5 arrays [76]. When storage system technology is not advanced sufficiently for a particular application's requirements, the existence of detailed models of storage system behavior enables the application to take direct responsibility for satisfying its needs.

4.7 Exploiting idle processing power

With processor performance increasing and costs decreasing, as a result of low-cost microprocessors and microcontrollers, there will be a profusion of computing power thorughout the computing system. Substantial additional processing power will be available not only in end-user processors but also in server machines, storage devices and subsystems, and network devices. Putting this additional processing power to work may be a powerful tool for meeting the strategic goals of I/O systems [21, 27]. The computing power is local to the devices and can take advantage of specialized and real-time knowledge. Subsystem processing can be used for on-the-fly modeling of alternative storage configurations and dynamic reconfiguration. It can also be used to execute more complex, application-provided data operations and algorithms for resource scheduling, prefetching, and caching. Subsystems are also likely to provide compression, search, timely delivery (for example, of continuous media), and format translation. There are also opportunities for tight coupling with network protocols. New directions in networking research, notably the ability to execute functions in network devices, might be put to good use for meeting end-to-end application requirements from storage access [74, 75]. In addition to exploiting the cycles of these idle machines in parallel and distributed applications, storage systems can exploit idle cycles or memory resources for global memory management [18, 24, 29].

4.8 Application influence over operating system policies

Although it is valid to say that operating systems were invented to hide storage access and management from applications, their handling of storage systems is more generic than is appropriate for many applications. Operating systems provide an uninterrupted, basic level of service to all applications, and thus code stability and generality are paramount. This also means that operating system models of storage device and subsystem capabilities are often obsolete, and their customization to application service requirements is usually minimal or total; that is, they may give the application the choice of taking either no responsibility or full responsibility for device access and management. Consequently, demanding applications such as database management systems prefer to take total responsibility [67]. In the analogous case of the network interface, there is also much interest in allowing applications to access devices directly and bear responsibility for management [6, 46, 72].

An important recent approach to application customization in operating systems can also be applied to storage systems. This is the use of mechanisms that allow applications to help operating systems make critical policy decisions [3, 22]. For example, applications and storage systems can

cooperate to provide optimizations specific to their access and management needs, such as better memory management, reduced data copying, and parallel execution of simple filtering functions near the data [28, 49].

5 Concluding Remarks

Storage systems are a large and important component of computing systems, and they will continue to play a vital role in the forseable future. The access gap shows no imminent sign of vanishing, and thus continuing research into storage I/O will be essential to reap the full benefit from the advances occurring in many other areas of computer science. In the storage I/O area, we have identified a few strategic research thrusts, which we summarize here:

First, the techniques developed across the wide range of computing domains involving storage management need to be collected into a coherent, reusable body of understanding, and brought to bear as a group upon the data employed by emerging applications. Such data is becoming increasingly structured and applications ever more interoperable and integrated; both are raising new challenges and new opportunities to exploit additional information in the storage system.

Second, expensive, complicated, and important storage-management functions, which were long left to users and administrators, should be made the responsibility of the storage subsystems. Such functions should be made markedly more dynamic and adaptive, so that they can cope automatically with changing needs.

Third, in customizing storage management to increasingly demanding applications, classes of service for storage should be identified, and integrated class resource scheduling should be developed.

Finally, the technological limitations of current storage devices are sorely felt. New device technologies, as well as corresponding software and management techniques, should be sought and rapidly adopted to expand the trade-offs available to customers, and to reduce (or better cope with) the access gap between the performances of main and secondary memories.

Acknowledgments. We wish to thank Anna Karlin, Edward Lee, David Patterson, Bernard O'Lear, A. L. Narasimha Reddy, Daniel Reed, Liuba Shrira, and David Womble for several helpful comments and suggestions.

References

[1] Acharya, A., Uysal, M., Bennett, R., Mendelson, A., Beynon, M., Hollingsworth, J. K., Saltz, J., and Sussman, A. Tuning the performance of I/O intensive parallel applications. In *Fourth Workshop on Input/Output in Parallel and Distributed Systems* (Philadelphia, May 1996), pp. 15–27.

[2] Baker, M. G., Hartman, J. H., Kupfer, M. D., Shirriff, K. W., and Ousterhout, J. K. Measurements of a distributed file system. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles* (October 1991), pp. 198–212.

[3] Bershad, B. N., Savage, S., Pardyak, P., Sirer, E. G., Fiuczynski, M., Becker, D., Eggers, S., and Chambers, C. Extensibility, safety and performance in the SPIN operating system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (Copper Mountain, Colorado, December 1995).

[4] Berson, S., Golubchik, L., and Muntz, R. R. Fault tolerant design of multimedia servers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1995), pp. 364–375.

[5] Brustoloni, J. C., and Steenkiste, P. Effects of buffering semantics on I/O performance. In *Proceedings of the 1996 OSDI Conference* (1996).

[6] Buzzard, G., Jacobson, D., Mackey, M., Marovich, S., and Wilkes, J. An implementation of the Hamlyn sender-managed interface architecture. In *Proceedings of the 1996 OSDI Conference* (Seattle, WA, October 1996), Usenix Association, Berkeley, CA.

[7] Cao, P., Felten, E. W., Karlin, A., and Li, K. Implementation and performance of integrated application-controlled caching, prefetching and disk scheduling. *ACM Transactions on Computer Systems* (to appear). An earlier version available as Technical Report CS–TR–94–493, Princeton University.

[8] Carey, M. J., DeWitt, D. J., Franklin, M. J., Hall, N. E., McAuliffe, M. L., Naughton, J. F., Schuh, D. T., Solomon, M. H., Tan, C. K., Tsatalos, O. G., White, S. J., and Zwilling, M. J. Shoring up persistent applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1994), pp. 383–394.

[9] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., and Patterson, D. A. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys 26*, 2 (June 1994), 145–185.

[10] Chen, S., and Towsley, D. A queueing analysis of RAID architectures. Tech. Rep. COINS 91–71, University of Massachusetts, Department of Computer and Information Science, University of Massachusetts, September 1991.

[11] Chiang, Y.-J., Goodrich, M. T., Grove, E. F., Tamassia, R., Vengroff, D. E., and Vitter, J. S. External-memory graph algorithms. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, CA, January 1995), pp. 139–149.

[12] Corbett, P., Feitelson, D., Fineberg, S., Hsu, Y., Nitzberg, B., Prost, J.-P., Snir, M., Traversat, B., and Wong, P. Overview of the MPI-IO parallel I/O interface. In *IPPS '95 Workshop on Input/Output in Parallel and Distributed Systems* (April 1995), pp. 1–15.

[13] Corbett, P. F., and Feitelson, D. G. The Vesta parallel file system. *ACM Transactions on Computer Systems 14*, 3 (August 1996), 225–264.

[14] Corbett, P. F., Feitelson, D. G., Prost, J.-P., Almasi, G. S., Baylor, S. J., Bolmarcich, A. S., Hsu, Y., Satran, J., Snir, M., Colao, R., Herr, B., Kavaky, J., Morgan, T. R., and Zlotek, A. Parallel file systems for the IBM SP computers. *IBM Systems Journal 34*, 2 (January 1995), 222–248.

[15] Cormen, T. H., and Colvin, A. ViC*: A preprocessor for virtual-memory C*. Tech. Rep. PCS–TR94–243, Dept. of Computer Science, Dartmouth College, November 1994.

[16] Cormen, T. H., and Wisniewski, L. F. Asymptotically tight bounds for performing BMMC permutations on parallel disk systems. In *Proceedings of the Fifth Symposium on Parallel Algorithms and Architectures* (June 1993), pp. 130–139.

[17] Curewitz, K., Krishnan, P., and Vitter, J. S. Practical prefetching via data compression. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (May 1993), pp. 257–266.

[18] Dahlin, M., Wang, R., Anderson, T., and Patterson, D. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the 1996 OSDI Conference* (November 1994).

[19] de Jonge, W., Kaashoek, M. F., and Hsieh, W. C. The logical disk: A new approach to improving file systems. In *Proc. of 14th ACM Symp. on Operating Systems Principles* (December 1993).

[20] del Rosario, J. M., and Choudhary, A. High performance I/O for parallel computers: Problems and prospects. *IEEE Computer 27*, 3 (March 1994), 59–68.

[21] Drapeau, A. L., Shirriff, K. W., Hartman, J. H., Miller, E. L., Sesha, S., Katz, R. H., Lutz, K., Patterson, D. A., Lee, E. K., Chen, P. M., and Gibson, G. A. Raid-ii: A high-bandwidth network file server. In *Proc. 21st Annual International Symposium on Computer Architecture* (April 1994).

[22] Engler, D. R., Kaashoek, M. F., and O'Toole, Jr., J. Exokernel: An operating system architecture for application-level resource management. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (Copper Mountain, CO, December 1995).

[23] English, R. M., and Stepanov, A. A. Loge: a self-organizing storage device. In *Proceedings of the USENIX Winter'92 Technical Conference* (San Francisco, CA, January 1992), pp. 237–251.

[24] Feeley, M. J., Morgan, W. E., Pighin, F. P., Karlin, A. R., Levy, H. M., and Thekkath, C. A. Implementing global memory management in a workstation cluster. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (Copper Mountain, Colorado, December 1995).

[25] Ferrari, D. Client requirements for real-time communication services. *IEEE Communications Magazine 28*, 11 (Nov. 1990), 65–72.

[26] Gelb, J. P. System managed storage. *IBM Systems Journal 28*, 1 (1989), 77–103.

[27] Gibson, G. A., Nagle, D. P., Amiri, K., Chang, F. W., Feinberg, E., Lee, H. G. C., Ozceri, B., Riedel, E., and Rochberg, D. A case for network-attached secure disks. Tech. Rep. CMU–CS–96–142, Carnegie-Mellon University, October 1996.

[28] Gibson, G. A., Stodolsky, D., Chang, P. W., Courtright II, W. V., Demetriou, C. G., Ginting, E., Holland, M., Ma, Q., Neal, L., Patterson, R. H., Su, J., Youssef, R., and Zelenka, J. The Scotch parallel storage systems. In *Proceedings of 40th IEEE Computer Society International Conference* (San Francisco, Spring 1995), pp. 403–410.

[29] Golding, R., Bosch, P., Staelin, C., Sullivan, T., and Wilkes, J. Idleness is not sloth. In *Proceedings of Winter USENIX Technical Conference* (Jaunary 1995), Usenix Association, Berkeley, CA, pp. 201–212.

[30] Golding, R., Shriver, E., Sullivan, T., and Wilkes, J. Attribute-managed storage. In *Workshop on Modeling and Specification of I/O* (San Antonio, TX, October 1995).

[31] Griffioen, J., and Appleton, R. Reducing file system latency using a predictive approach. In *Proc. of 1994 Summer USENIX Conf.* (Boston, MA, 1994).

[32] Holland, M., Gibson, G. A., and Siewiorek, D. P. Architectures and algorithms for on-line failure recovery in redundant disk arrays. *Journal of Distributed and Parallel Databases 2*, 3 (July 1994), 295–335.

[33] Hurson, A., Pakzad, S. H., and bing Cheng, J. Object-oriented database management systems: Evolution and performance issues. *IEEE Computer* (February 1993).

[34] IEEE Storage System Standards Working Group, Project 1244. *Reference Model for Open Storage Systems Interconnection*, September 1994. See also http://www.arl.mil/IEEE/ssswg.html.

[35] Jacobson, D. M., and Wilkes, J. Disk scheduling algorithms based on rotational position. Tech. Rep. HPL–CSP–91–7, Hewlett-Packard Laboratories, Palo Alto, CA, 24th February (revised 1st March) 1991.

[36] Kimbrel, T., Tomkins, A., Patterson, R. H., Bershad, B., Cao, P., Felten, E. W., Gibson, G., Karlin, A. R., and Li, K. A trace-driven comparison of algorithms for parallel prefetching and caching. In *Proceedings of the 1996 OSDI Conference* (1996).

[37] Kohl, J. T., and Staelin, C. HighLight: using a log-structured file system for tertiary storage management. In *Proceedings of the Winter USENIX Technical Conference* (San Diego, CA, January 1993), Usenix Association, Berkeley, CA, pp. 435–447.

[38] Kotz, D. Disk-directed I/O for MIMD multiprocessors. In *Proceedings of the 1994 Symposium on Operating Systems Design and Implementation* (November 1994), pp. 61–74. Updated as Dartmouth TR PCS–TR94–226 on November 8, 1994.

[39] Kotz, D., and Ellis, C. S. Prefetching in file systems for MIMD multiprocessors. *IEEE Transactions on Parallel and Distributed Systems 1*, 2 (April 1990), 218–230.

[40] Kotz, D., and Ellis, C. S. Practical prefetching techniques for multiprocessor file systems. *Journal of Distributed and Parallel Databases 1*, 1 (January 1993), 33–51.

[41] Lee, E. K., and Katz, R. H. An analytic performance model of disk arrays. In *Proceedings of SIGMETRICS* (1993), pp. 98–109.

[42] Lee, E. K., and Katz, R. H. The performance of parity placements in disk arrays. *IEEE Transactions on Computers 42*, 6 (June 1993), 651–664.

[43] Lee, E. K., and Thekkath, C. A. Petal: Distributed virtual disks. In *Proceedings of the 1996 ASPLOS Conference* (1996).

[44] Li, Z., Reif, J. H., and Gupta, S. K. S. Synthesizing efficient out-of-core programs for block recursive algorithms using block-cyclic data distributions. In *Proceedings of the 1996 International Conference on Parallel Processing* (August 1996).

[45] Liskov, B., Maheshwari, U., and Ng, T. Partitioned garbage collection of a large stable heap. In *Proceedings of IWOOOS 1996* (Seattle, WA, 1996).

[46] Maeda, C., and Bershad, B. Protocol service decomposition for high-performance networking. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles* (December 1993), pp. 244–255.

[47] Miller, E. L., and Katz, R. H. An analysis of file migration in a Unix supercomputing environment. In *Proceedings of the Winter USENIX Technical Conference* (San Diego, CA, January 1993), Usenix Association, Berkeley, CA, pp. 421–433.

[48] Mogi, K., and Kitsuregawa, M. Dynamic parity stripe reorganizations for RAID5 disk arrays. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems* (September 1994), pp. 17–26.

[49] Mosberger, D., and Peterson, L. L. Making paths explicit in the Scout operating system. In *Proceedings of the 1996 OSDI Conference* (1996).

[50] Mowry, T. C., Demke, A. K., and Krieger, O. Automatic compiler-inserted I/O prefetching for out-of-core applications. In *Proceedings of the 1996 OSDI Conference* (1996).

[51] Noble, B., Price, M., and Satyanarayanan, M. A programming interface for application-aware adaptation in mobile computing. In *Proceedings of the Second USENIX Symposium on Mobile & Location-Independent Computing* (Ann Arbor, MI, April 1995).

[52] O'Toole, J., and Shrira, L. Opportunistic log: Efficient installation reads in a reliable object server. In *Proc of First Usenix Conference On Operating System Design and Implementation* (Monterey, CA, 1994).

[53] Pasquale, J., Anderson, E., and Muller, P. K. Container shipping: Operating system support for I/O-intensive applications. *IEEE Computer* (March 1994).

[54] Patterson, R. H., Gibson, G. A., Ginting, E., Stodolsky, D., and Zelenka, J. Informed prefetching and caching. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (December 1995), pp. 79–95.

[55] Purakayastha, A., Ellis, C. S., and Kotz, D. ENWRICH: a compute-processor write caching scheme for parallel file systems. In *Fourth Workshop on Input/Output in Parallel and Distributed Systems* (May 1996), pp. 55–68.

[56] Reddy, A. N., and Wyllie, J. C. I/O issues in a multimedia system. *IEEE Computer* (March 1994).

[57] Rosenblum, M., and Ousterhout, J. K. The design and implementation of a log-structured file system. *ACM Trans. on Computer Sys. 10*, 1 (Feb. 1992).

[58] Ruemmler, C., and Wilkes, J. Disk shuffling. Tech. Rep. HPL–91–156, Hewlett-Packard Laboratories, Palo Alto, CA, October 1991.

[59] Ruemmler, C., and Wilkes, J. UNIX disk access patterns. In *Proceedings of the Winter USENIX Technical Conference* (San Diego, CA, January 1993), pp. 405–420.

[60] Ruemmler, C., and Wilkes, J. An introduction to disk drive modelling. *IEEE Computer 3*, 27 (March 1994), 17–28.

[61] Savage, S., and Wilkes, J. AFRAID— a frequently redundant array of independent disks. In *Proceedings of the 1996 Winter USENIX Conference* (January 1996), pp. 27–39.

[62] Seamons, K. E., Chen, Y., Jones, P., Jozwiak, J., and Winslett, M. Server-directed collective I/O in Panda. In *Proceedings of Supercomputing '95* (December 1995).

[63] Seltzer, M., Chen, P., and Ousterhout, J. Disk scheduling revisited. In *Proceedings of Winter USENIX Technical Conference* (January 1990), p. 313 323.

[64] Shrira, L., Liskov, B., Castro, M., and Adya, A. How to scale transactional storage systems. In *Proceedings of SIGOPS European Workshop on Operating System Support for World Wide Applications* (Connemara, Ireland, 1996).

[65] Shriver, E. A. M., and Wisniewski, L. F. An API for choreographing data accesses. Tech. Rep. PCS–TR95–267, Dartmouth College Department of Computer Science, October 1995.

[66] Sienknecht, T. F., Friedrich, R. J., Martinka, J. J., and Friedenbach, P. M. The implications of distributed data in a commercial environment on the design of hierarchical storage y management. *Performance Evaluation 20*, 1–3 (May 1994), 3–25.

[67] Stonebraker, M. Operating system support for database management. *Communications of the ACM 7*, 24 (July 1981).

[68] Tait, C. D., and Duchamp, D. Detection and exploitation of file working sets. In *Proceedings of the 11th International Conference on Distributed Computing Systems* (Arlington, TX, 1991), IEEE Computer Society, Washington, DC, pp. 2–9.

[69] Thakur, R., Choudhary, A., Bordawekar, R., More, S., and Kuditipudi, S. Passion: Optimized I/O for parallel applications. *Computer* (June 1996), 70–78.

[70] Vengroff, D. E., and Vitter, J. S. I/O-efficient computation: The TPIE approach. In *Proceedings of the Goddard Conference on Mass Storage Systems and Technologies* (College Park, MD, September 1996), NASA Conference Publication 3340, Volume II, pp. 553–570.

[71] Vitter, J. S., and Shriver, E. A. M. Algorithms for parallel memory, I: Two-level memories and II: Hierarchical multilevel memories. *Algorithmica 12*, 2/3 (August and September 1994), 110–169.

[72] von Eicken, T., Culler, D. E., Goldstein, S. C., and Schauser, K. E. Active messages: A mechanism for integrated communication and computation. In *Proceedings of the 19th ACM International Symposium on Computer Architecture* (May 1992), pp. 256–266.

[73] Vongsathorn, P., and Carson, S. D. A system for adaptive disk rearrangement. *Software—Practice and Experience 20*, 3 (March 1990), 225–242.

[74] Wetherall, D., and Tennenhouse, D. The ACTIVE IP option. In *Proceedings of the 7th ACM SIGOPS European Workshop* (Connemara, Ireland, September 1996).

[75] Wilkes, J. DataMesh research project, phase 1. In *Proceedings of the USENIX File Systems Workshop* (May 1992), pp. 63–69.

[76] Wilkes, J., Golding, R., Staelin, C., and Sullivan, T. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems 14*, 1 (February 1996), 108–136.

[77] Wilkes, J., and Stata, R. Specifying data availability in multi-device file systems. In *Operating Systems Review* (1991), vol. 25 of *Position paper for 4th ACM SIGOPS European Workshop*, pp. 56–59.

[78] Worthington, B. L., Ganger, G. R., Patt, Y. N., and Wilkes, J. On-line extraction of SCSI disk drive parameters. In *Proceedings of SIGMETRICS'95* (Ottawa, Canada, May 1995), pp. 146–156.

[79] Yong, V., Naughton, J., and Yu, J. Storage reclamation and reorganization in client–server persistent object stores. In *In Proceedings of Data Engineering Conference* (Houston, TX, 1994).