Storage Hierarchy Management for Scientific Computing

by
Ethan Leo Miller

Sc. B. (Brown University) 1987
M.S. (University of California at Berkeley) 1990

A dissertation submitted in partial satisfaction of the requirements for the degree of
Doctor of Philosophy
in
Computer Science
in the
GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Randy H. Katz, Chair
Professor Michael Stonebraker
Professor Phillip Colella

1995

Storage Hierarchy Management for Scientific Computing

Abstract

Scientific computation has always been one of the driving forces behind the design of computer systems. As a result, many advances in CPU architecture were first developed for high-speed supercomputer systems, keeping them among the fastest computers in the world. However, little research has been done in storing the vast quantities of data that scientists manipulate on these powerful computers.

This thesis first characterizes scientists' usage of a multi-terabyte tertiary storage system attached to a high-speed computer. The analysis finds that the number of files and average file size have both increased by several orders of magnitude since 1980. The study also finds that integration of tertiary storage with secondary storage is critical. Many of the accesses to files stored on tape could have easily been avoided had scientists seen a unified view of the mass storage hierarchy instead of the two separate views of the system studied. This finding was a major motivation of the design of the RAMA file system.

The remainder of the thesis describes the design and simulation of a massively parallel processor (MPP) file system that is simple, easy to use, and integrates well with tertiary storage. MPPs are increasingly commonly used for scientific computation, yet their file systems require great attention to detail to get acceptable performance. Worse, a program that performs well on one machine may perform poorly on a similar machine with a slightly different file system. RAMA solves this problem by pseudo-randomly distributing data to a disk attached to each processor, making performance independent of program usage patterns. It does this without sacrificing the high performance that scientific users demand, as shown by simulations comparing the performance of RAMA and a striped file system on both real and synthetic benchmarks. Additionally, RAMA can be easily integrated with tertiary storage systems, providing a unified view of the file system spanning both disk and tape systems. RAMA's ease of use and simplicity of design make it an ideal choice for the massively parallel computers used by the scientific community.

Table of Contents

List of Figures

List of Tables

1 Introduction

Scientific computing has always been one of the driving forces behind computer architecture. The first computers were designed to compute weapon trajectories, and weather forecasting was an early user of many cycles. Since then, computers have become significantly faster, permitting ever-larger scientific computations that consume and produce ever-increasing amounts of data. Today's workstations are faster than the supercomputers of fifteen years ago, and processor speeds continue to double every three years. The introduction of parallel processors has further accelerated the increase in processor speed. However, data storage bandwidth has not kept pace because of mechanical limitations — disk rotation speed, for example, has barely doubled in two decades. While disks and other storage devices have not seen great increases in speed, their capacity per unit volume has increased dramatically. Today's disk drives store more data than those of ten years ago, yet they occupy less than 10% of the volume of the earlier drives. Increasingly, storage device bandwidth is presenting a bottleneck to scientific computation. Moving data between disks and CPUs is one concern retarding the acceptance of massively parallel processors. Archival storage presents an additional bottleneck to scientific computation. Modern supercomputer centers can only afford storage systems with the capacity and cost of tape, yet scientists want to access this storage with the bandwidth and latency disk provides.

1.1. Thesis Statement

The objective of this dissertation is to address two issues in the design of mass storage systems for scientific computation: characterization of modern multi-terabyte storage systems, and high bandwidth file systems for massively parallel processors. These issues cannot simply be addressed as isolated problems, however; both high-bandwidth storage and massive information repositories must be part of an integrated storage environment for scientific computing. Building a coherent file system from many individual storage systems is as difficult a problem as is building each individual component. Thus, this thesis also addresses issues of designing an integrated storage system for supercomputing. While performance and capacity are always recognized as important metrics of storage systems, ease of use is often neglected. Ideally, users should see a single file system entity, in contrast to current systems composed of several loosely-connected file systems between which files must be explicitly moved by users. Additionally, current parallel file systems such as Vesta [18] and the Intel Paragon file system [27] require a user to supply file layout directives for the file system to provide maximum performance. These directives are not necessary; however, the file system's performance may suffer without them. Future massive storage systems supporting scientific computing must be easy to use as well as high-bandwidth and high-capacity.

The first third of the dissertation presents a detailed analysis of a modern tertiary storage system, providing a basis for the design and analysis of new algorithms for moving data between disks and tertiary storage devices such as tapes. The most recent extensive study of a similar system was done around 1980 and reported in [80] and [81]. However, both access patterns to mass storage and the hardware used to build

them have since changed dramatically. The introduction of new storage devices, such as inexpensive high-capacity tape robots and optical disk jukeboxes has given scientists more ways to store their data. At the same time, the rapid increase in supercomputer speeds has allowed scientists to generate hundreds of thousands of files per year averaging 25 megabytes per file. These files are two orders of magnitude larger than the files studied in [80]. Clearly, this data must be stored on a medium less expensive than disk. Modern supercomputer centers store this data on tapes; however, their storage system designs and algorithms are *ad hoc*, since there is little research on modern systems on which to base their choices. This thesis presents such an analysis, laying the groundwork for future research on file migration algorithms and mass storage systems designs.

The remaining two thirds of the dissertation focus on file systems for scientific computing on massively parallel processors (MPPs), which are both difficult to design and hard to use efficiently. Current MPP file systems can provide good performance, but do so at the cost of ease of use. In a traditional uniprocessor, files are stored on disks attached directly to the computer or on remote machines reachable by a network. The bandwidth in such storage systems need only scale with the speed of the single processor, while the connection between the disk and the processor's memory need only be fast enough to handle the data from several disks at the same time. For large multiprocessors, however, *each* processor might consume data at a rate comparable to that for an entire traditional computer. Furthermore, the available disk bandwidth must be scalable — if more processors are added, more bandwidth is necessary. MPPs with hundreds of processors require secondary storage bandwidth of several hundred megabytes per second. While it may be possible to build an single connection between hundreds of disks and hundreds of processors, it makes more sense to build many lower-performance links instead. Recent advances in disk technology allow the storage of hundreds of megabytes on a disk smaller than a deck of playing cards, while modern high-speed MPP interconnection networks allow data to be stored further from its eventual destination with little performance penalty. The remaining two thirds of this dissertation proposes and analyzes a parallel file system combining these two advances. This file system, called RAMA, is well-suited for scientific applications that provides high bandwidth and eliminates the need for complex user directives. Additionally, this file system is designed to be integrated into a larger storage system including workstations and tertiary storage systems, in contrast to current parallel file systems that require users to explicitly move data from the outside world to the MPP.

1.2. Dissertation Outline

The thesis opens with the background material built upon by the rest of the dissertation. Chapter 2 first discusses the characteristics of secondary storage devices such as disk drives and tertiary storage devices such as magnetic tape, optical disk, and optical tape. Next, the chapter presents the basic file system concepts necessary for the following chapters, using the BSD Fast File System [54] and the Log Structured File System [74] as examples. The chapter then covers previous research in two areas: file migration and tertiary storage management, and parallel file systems. The survey of file migration discusses several studies of tertiary storage access patterns and proposals of migration algorithms. This research provides a good background for study of modern storage systems; however, the large increases in capacity and usage require a reexamination of the problem. Parallel file systems, on the other hand, are recent developments. The chapter mentions a half dozen different file systems, summarizing each one's strong and weak points. While the various parallel file systems each have different weaknesses, all share a common problem: they require system-specific input from the user for optimal performance.

The remainder of the thesis follows the methodology diagrammed in Figure 1-1. The file migration study in Chapter 3 yields several supercomputer file system requirements that traditional parallel systems do not satisfy. These requirements, along with the more standard demands on parallel file systems, drive the design of the RAMA file system. Simulation studies of RAMA's sensitivity and its performance relative to conventional parallel file systems may be used later to implement a high performance file system that both

meets the high-performance goals of supercomputer systems and provides a good interface with the rest of the computing environment.

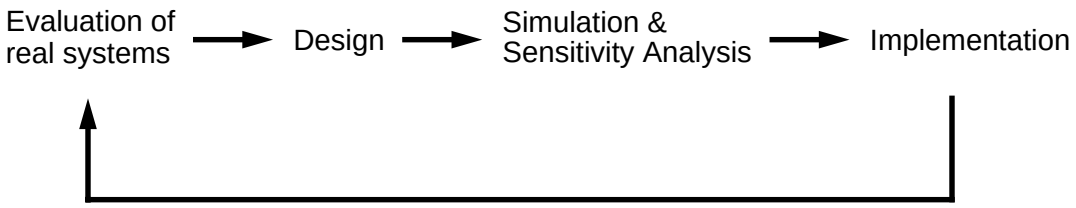

Figure 1-1. Thesis methodology.

This thesis follows a standard methodology for developing new systems. The analysis of existing supercomputer file migration systems in this dissertation combined with the analyses of parallel systems done elsewhere motivate the design of a new parallel file system — RAMA. Simulation is used to test this design's performance and sensitivity to changes in technology and design parameters. The information gained from these simulations will be used to refine the design for an actual implementation — part of the future work described in Chapter 8.

Chapter 3 presents the results of a trace-based study of a typical mass storage system and discusses its implications for file migration algorithm design. It first describes the system at the National Center for Atmospheric Research (NCAR), and then covers the gathering and processing of the access traces, which included references to almost one million files containing a total of over 25 terabytes of data. The bulk of the chapter analyzes the traces gathered at NCAR over a two year period. The analysis shows that files at NCAR fall into two main groups: half of the files are referenced two or fewer times, and most of the rest are referenced frequently. Additional references to a file are most likely within a day or two of previous access to that file, though poor integration in the NCAR storage system forced two tertiary storage references regardless of how close together the two accesses occurred. There was little difference between the access patterns for large and small files. Reference patterns did differ, however, between reads and writes. File read rates closely matched scientists' schedules, peaking during the work day and dropping off at night and on weekends. On the other hand, file write rates remained relatively constant throughout the day and the week, suggesting that the write rate is governed more by the computer's ability to generate new data that must be stored.

Several findings reported in Chapter 3 motivate the proposal of the RAMA file system, the focus of the rest of the thesis. First, storage integration is crucial to good file system performance. More than half of the references to tertiary storage could have been avoided by transparent access to tertiary storage; however, current parallel file systems are not well-integrated into supercomputer storage centers. Second, the storage of file metadata is becoming a problem. Many of the supercomputer file systems that allow transparent access to tertiary storage require all of the file system metadata and directories to remain on disk even when files are migrated to tape. Chapter 4 presents the RAMA parallel file system design, which is primarily motivated by these two considerations and a third: ease of use. RAMA uses pseudo-random distribution and reverse indices similar to cache tags to spread data around the disks of an MPP. Because RAMA uses reverse indices to find file blocks, it only keeps metadata for files actually on disk; metadata for migrated files may itself be migrated. This strategy also allows seamless integration of tertiary storage — if the reverse indices do not locate a file on disk, it can be retrieved from tape without manual intervention by the user. As a parallel file system, however, RAMA's major attraction is its performance. Its pseudo-random distribution probabilistically guarantees high bandwidth regardless of an application's access pattern. Unlike current MPP file

systems, RAMA is designed to achieve high bandwidth without extensive placement and usage information from the applications that use it, thus providing an easy-to-use parallel file system.

The remainder of the thesis describes a simulation of the RAMA file system and the results from experiments that use the simulator to predict RAMA's behavior under varying conditions. Chapter 5 discusses the methodology used to produce the performance figures reported in Chapters 6 and 7. It first describes the how the simulator is organized and the disk and network models it uses. The chapter next discusses the workload generators that generate reference streams to drive the simulator's model of RAMA. Several of the generators model real applications such as matrix decomposition, while others drive the simulation with synthetic access traces. Both real and synthetic reference streams will be used to generate the results reported in the next two chapters.

The sensitivities of RAMA to varying design and technological parameters are examined in Chapter 6. Advances in disk technology increase RAMA's bandwidth by allowing more data to go to or from disk. Faster networks, however, have little impact on the file system's bandwidth because it is currently disk-limited. This finding is encouraging since interconnection link speed was a serious performance bottleneck in earlier parallel file systems. RAMA's performance also varies for different design choices at a fixed technology point. While RAMA has fewer design parameters than other parallel file systems, the few design choices that must be made do affect file system performance. As with other file systems, though, the optimal choice for these parameters depends on the current technology. File system workload is the final factor affecting file system performance that Chapter 6 explores. Most of this thesis assumes that a supercomputer workload will use primarily large files. The last section of this chapter shows that RAMA will perform well under a small file workload as well, making RAMA suitable for use in an environment combining supercomputers and workstations.

In contrast to Chapter 6, which shows that RAMA will scale well with advancing technology and differing workloads, Chapter 7 demonstrates that pseudo-random distribution performs well when compared to standard striping. Moreover, striping performs well for some access patterns and poorly for others; an application must provide configuration information to the file system to distribute data for optimal performance. RAMA, on the other hand, achieves bandwidth close to that of the optimal striping layout, and maintains that level of performance across a wide range of access patterns for which the optimal striping arrangement varies widely. RAMA's performance is usually within 10% of the best performance possible from striping, and is a factor of 4 or more higher than the bandwidth attainable from a file system with a poorly-chosen stripe size. It is this combination of good performance and independence of performance from access pattern that make RAMA an attractive alternative to traditional striped parallel file systems.

The thesis finishes with Chapter 8, which summarizes the research reported in the dissertation and discusses avenues for future research. Since the area of storage, particularly massive storage systems for high-performance computing, is relatively unstudied, many opportunities for further research present themselves. This chapter discusses two such areas — the development of algorithms for file migration based on the analysis from Chapter 3 and other trace-driven studies, and the actual construction of the RAMA file system on a parallel processor with one or more disks per node. Addressing these research issues will further improve the computer science community's ability to support scientific research by providing better-performing, easier-to-use massive storage systems.

2 Background and Related Work

Supercomputer file systems have long been the subject of research, as users of large high-speed computers have always demanded the highest performance from all aspects of their systems. Computers used for scientific research employ many types of devices to provide the combination of high bandwidth and terabytes of storage. Storage systems are more than hardware, however. Modern high-speed computers need complex file systems for two reasons. First, these file systems must provide rapid access to the high bandwidth devices in the storage system, as supercomputers are capable of producing and consuming data at rates in excess of 50 MB per second [60]. Second, supercomputer file systems must migrate files between storage devices, placing unneeded data on slower but cheaper media and keeping active data on faster, more expensive media. With the advent of massively parallel processors (MPPs), however, file systems for scientific computing must address a third challenge — providing scalable high-speed file access to many processors at the same time.

This chapter will first describe the devices from which file systems are made. These include both secondary storage devices such as magnetic disks and tertiary storage devices including magnetic tapes, optical tapes, and optical disks. While secondary storage devices perform better than tertiary storage devices, they are more expensive per bit of data stored. The first section of this chapter will explore the differences between various devices and lay the foundation for discussing tradeoffs between cost and performance.

An overview of file system design comes next, providing a foundation for the discussion of file systems for scientific computation in this thesis. This section will cover some basic concepts and terminology of file system design, drawing examples from well-known systems such as Berkeley's Fast File System (FFS) and Log-Structured File System (LFS). The section will explain the data structures used by these file systems, and discuss the implications of these design choices. This discussion will be particularly useful for Chapter 4, which describes the design of the RAMA parallel file system.

This chapter will next discuss some previous work on file migration and tertiary storage systems. Mass storage systems have been used for nearly two decades, yet there are relatively few published papers describing them. Moreover, many of these studies are over ten years old. Since storage devices and usage patterns have changed greatly since then, these studies may have limited applicability to modern systems. However, they do provide a good base for analyzing modern mass storage systems.

The chapter concludes with an overview of work on parallel file systems for scientific computation. The recent increase in the number of massively parallel processors (MPPs) has resulted in the creation of several new parallel file systems. This section describes these systems as well a few older parallel file systems, detailing the strengths and weaknesses of each design.

2.1. Storage Devices

The first step in discussing storage systems is to describe the devices that such systems use. Figure 2-1 shows the "storage pyramid," which relates all the forms of storage to each other. Generally, capacity increases towards the bottom of the pyramid, while bandwidth is higher for storage at the top. Cost per byte is lowest for devices lower on the pyramid.

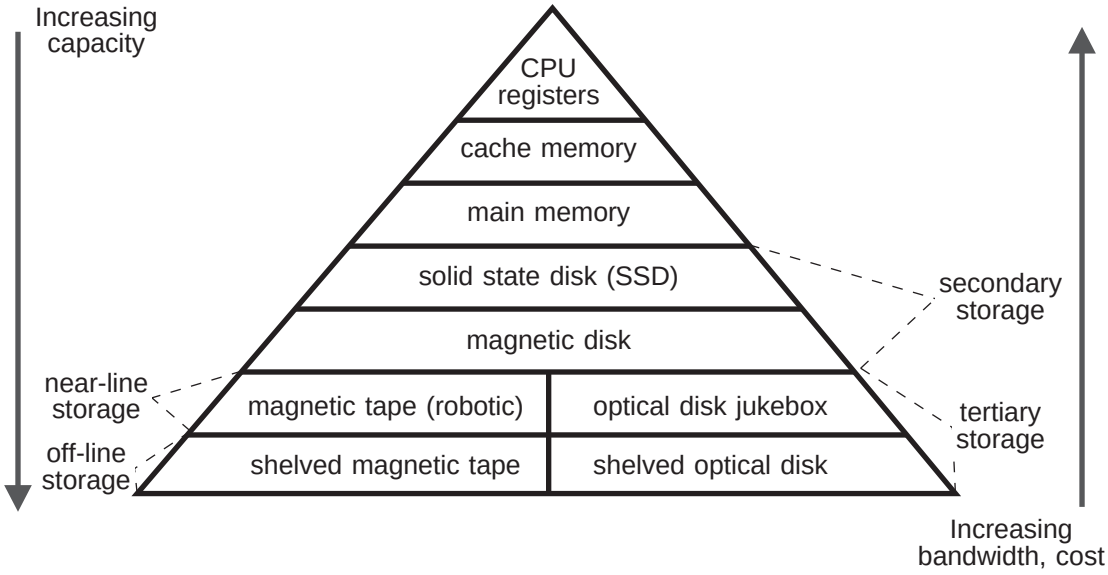

Figure 2-1. The storage pyramid.

This pyramid shows the range of storage devices from CPU registers through shelved magnetic tape and optical disk. Devices and media at the bottom of the pyramid are considered tertiary storage devices, and have the lowest cost per byte and the longest access times. Secondary storage, such as magnetic disk, has a somewhat higher cost per megabyte and shorter access time. Tertiary storage is further divided into offline and nearline storage. Offline storage is accessible only via human operator intervention, while nearline storage is robotically managed. As a result, nearline storage is usually faster than offline storage. However nearline storage is more expensive, as it is difficult to build storage for many thousands of tapes that is robotically-accessible.

Data storage devices are classified into one of two categories depending on whether the device has easily interchangeable media. Magnetic disk and solid state disk (SSD) are termed *secondary storage* devices, and require one device per storage medium. A magnetic disk drive is an integrated unit; it is impossible to switch media on today's disk drives. All other storage devices are called *tertiary storage* devices. Optical disks and magnetic and optical tapes are all tertiary storage devices, as a single read/write device can read any of a large number of storage media.

Tertiary storage is further broken down into *off-line* and *near-line* (or *robo-line*, as used in [46]) storage. Both types of tertiary storage use the same media, but they access those media in different ways. Near-line storage is available with very little latency, as the media are loaded by robots. The StorageTek Automated Cartridge System 4400 [48] is an example of such a system. Each tape silo stores 6,000 IBM 3490 (1/2 inch) tapes. Two robotic arms inside the silo may pick any tape and insert it into one of several IBM 3490 tape drives. Once the tape has been placed into a drive, the robotic arm is free to move other tapes. These arms can rapidly move a tape between drive and storage slot; an arm in the STK 4400 requires an average of 10

- 20 seconds to pick a single tape. The entire process of selecting a tape, moving it between its slot and the tape drive, and transferring data is automatic. No human need intervene in the process, keeping latencies low.

Off-line storage, on the other hand, is not accessible without human intervention. A mass storage center such as the National Center for Atmospheric Research (NCAR) [65] stores 25 terabytes or more, but only has sufficient tape silos for a fraction of that data. The remainder of the media are stored on rows of shelves in the data center and retrieved by the mass storage system's operators. Off-line accesses are considerably slower than near-line accesses for two reasons. First, human operators simply cannot move as quickly as robot arms. Second, humans may make mistakes both in picking the incorrect cartridge and in replacing a cartridge in the incorrect slot. For these reasons, off-line storage is slower than near-line storage. Nevertheless, data centers must store data off-line because it costs less to build shelves than buy robotic silos.

The remainder of this section discusses various types of secondary and tertiary storage. Magnetic disk is the only type of secondary storage covered; while solid-state disk (SSD) is used in many mass storage systems, its characteristics are those of a large array of dynamic RAM. Tertiary storage media are based either on magnetic or optical technologies. Since scientific computation centers use magnetic tape for tertiary storage far more than either optical tape or optical disk, it will be covered in more detail. Similarly, robots exist to manipulate both magnetic tape and optical disk, but tape robots dominate because of the prevalence of magnetic tape over optical technologies. The section concludes with a discussion of possible future directions for storage technologies including holographic optical storage.

2.1.1. Magnetic Disk

Magnetic disks have been used to store data for more than three decades, and their dominance of secondary storage seems to be secure for at least another decade. Over the past decade, disk capacities have been steadily increasing even as disks have become smaller. Disk performance, however, has not kept up with increases in data density.

The components of a typical magnetic disk are shown in Figure 2-2. While the original disks had platters that were or more 14 inches across, modern commodity disk drives are 5.25 or 3.5 inches across, and 2.5 inch diameter drives are becoming more common. Nonetheless, basic disk drive design has changed little in decades. Advancing technology is improving drive characteristics, however, as summarized in Table 2-1. Disk capacity and data density are increasing most rapidly — 27% per year [11], doubling every three years. Transfer rate is increasing at only 22% annually, however, and even this rate of increase may be difficult to sustain as rotation rates plateau. Average seek times are improving even more slowly, as they drop only 8% annually.

A single disk I/O request goes through four phases: switch to the correct read/write head, seek to the correct track, rotate to the proper sector, and transfer data. These phases may be overlapped, reducing the total time needed for a single I/O. Since a drive has one read/write head for each surface, it must switch to the appropriate one for each I/O. This is a rapid process, though, since switching heads is an electronic action rather than a physical one. Seek and rotational latency, however, are physical delays. The disk actuator must move the read/write head to the correct cylinder, taking 1 to 25 ms depending on the disk and the distance the head must travel. After the head reaches the desired track, the drive must wait for the sector to be transferred to rotate under the head. At this point, the data may be transferred.

This description of a disk's operation is simplified; modern disks use caches and other techniques to reduce the time necessary to satisfy an I/O request. [75] contains a more detailed description of modern disk operations.

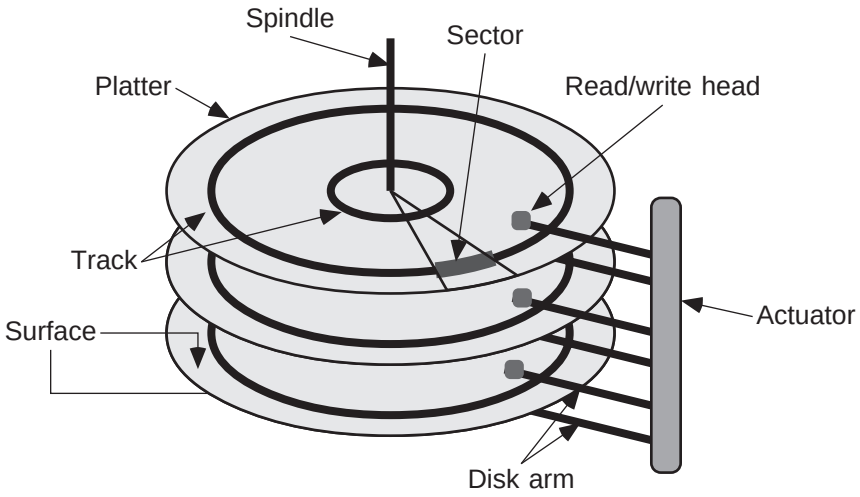

Figure 2-2. Components of a typical disk drive.

This diagram shows the components of a disk drive and other drive-related terminology. The *sector* is the basic unit of data storage, and holds 256 - 4,096 bytes, depending on how the disk is formatted. Read and write operations always occur on whole sectors. Each *track* has many sectors along its circumference. A *platter* is a single disk covered with metal oxide. Each platter has two *surfaces* (top and bottom in this diagram) that hold thousands of concentric tracks of data apiece. The set of corresponding tracks on every surface in the drive is termed a *cylinder.*

	1993 typical values	Historical rate of improvement
Areal density	50 - 150 Mbits/sq. inch	27% per year
Linear density	40,000 - 60,000 bits/inch	13% per year
Inter-track density	1,500 - 3,000 tracks/inch	10% per year
Capacity (3.5'' form factor)	100 - 2000 MB	27% per year
Transfer rate	3 - 4 MB/s	22% per year
Seek time	7 - 20 ms	8% per year
Rotation rate	120 rotations/s	4% per year

Table 2-1. Trends in disk technology.

While magnetic disks' capacities are increasing rapidly, their performance is increasing more slowly. Transfer rate is proportional to the product of linear density and rotation rate, neither of which is improving rapidly.

The data in this table (with the exception of rotation rate) is taken from Table 2 in [11].

2.1.2. Magnetic Tape

Magnetic tape, like magnetic disk, is a rather old technology. Tapes have long been used to hold archival copies of data on disk. This is done for two reasons: keeping an extra copy of disk data in case of a disk failure, and storing excess data that does not fit on disk. Magnetic tape is ideally suited for these uses, since

tapes have very low cost per megabyte of stored data (see Table 2-2). However, tapes are poorly suited for holding active data because they are linear media and have average seek times longer three orders of magnitude slower than disk seek times.

Technology	Capacity (MB)	Media cost	Density (Mb/in^2)	Transfer rate (KB/s)	Access time (1/3 full seek)	Exchange time	Media life (passes)
Longitudinal tapes							
Reel-to-reel (1/2")	140	$5	0.11	549	minutes	minutes	∞
Cartridge (1/4")	150	$15	1.25	92	minutes	minutes	∞
IBM 3490E (1/2")	800	$10	1.74	6,000	15 s	10 s	∞
Helical scan tape							
VHS (1/2")	15,000	$29	?	4,000	45 s	6 s	500
8mm (video) [4]	4,600	$14	70.56	492	45 s	100 s	500
4mm DAT [62,84]	1,300	$8	114.07	183	20 s	55 s	500
19mm (DD-2)	25,000	$140	46.00	15,000	15 s	5 s	1,000
Optical tape							
35mm (CREO)	10^6	$5,000	224.00	3,000	30 s	mins	∞
Optical disk							
Kodak (14")	3,200	$500	296.33	1,000	100+ ms	7 s	∞

Table 2-2. Characteristics of various tertiary storage technologies.

This table summarizes the characteristics of several tertiary storage technologies, including both magnetic tape and optical storage. The figure for media life has different meaning for magnetic and optical technologies. For magnetic tape technologies, it refers to the maximum number of passes the tape may make past the read/write heads, regardless of whether the tape is read or written. Optical technologies, on the other hand, are write-once. Thus, the media life figure refers to the maximum number of times the media may be read. Exchange time includes the time to unload a medium currently in the device and load a new one. Thus, load and unload time each require half of this time.

This table is updated from similar data reported in [46].

There are two basic types of magnetic tape mechanisms: *longitudinal* (or *linear*) and *helical scan* [4,94]. In longitudinal tape drives, the tape passes by stationary read and write heads, as shown in Figure 2-3. Data is written on the tape in one or more *tracks*, each of which runs the length of the tape. The tape drive's performance is determined by how rapidly the tape moves across the heads and how many bytes are written per inch of tape (*linear density*). Tape capacity is found by multiplying the linear density by the length of the tape. Loading a tape into a longitudinal tape drive is a relatively quick process, typically requiring a few seconds.

Helical scan tape drives, in contrast, wrap the tape around a rotating cylinder with several read and write heads on it as shown in Figure 2-3. This allows the drive to advance the tape more slowly than for longitudinal tape while maintaining high bandwidth, since the heads rotate rapidly past the slow-moving tape. While helical scan tapes can provide higher density than longitudinal tapes, they do have drawbacks. First, helical scan tapes are either slow or expensive. Both 8mm and 4mm technologies feature inexpensive drives and media, but are hampered by low transfer rates and very long load times. The load time for an Exabyte 8mm tape drive, for example, is 100 seconds [26]. 19mm and VHS technologies, on the other hand, have

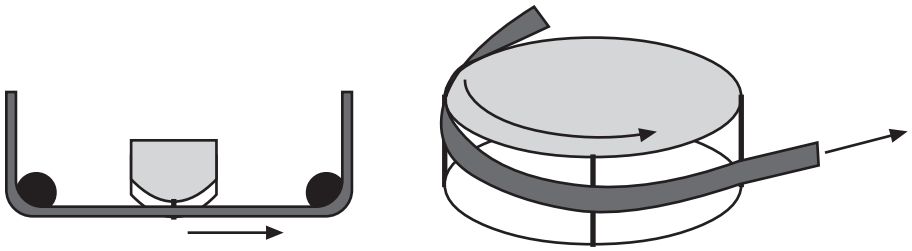

Figure 2-3. Longitudinal and helical scan tape technologies.

This diagram compares the tape path through longitudinal and helical scan tape drives. The longitudinal system on the left has a considerably simpler tape path, as the tape contacts the read/write head at only one point. The helical scan system on the right is more complex, with the tape wound around a cylinder with multiple read/write heads on its surface.

high transfer rates and short load times. However, these two technologies require expensive drives. In addition, helical scan tapes and read/write heads suffer from greater wear problems than do longitudinal tape because of the greater contact area between tape and heads.

Table 2-2 summarizes the characteristics of several tape technologies. These are split into helical scan and longitudinal tapes. The most common tape technology in supercomputer centers today is IBM 3490 tapes. Each full-length tape cartridge holds 400 MB, depending on the tape drive model, and data can be read or written at 3-6 MB per second. These tapes are commonly used because the technology has existed since before helical scan technology became widespread and supercomputer data centers are reluctant to switch to new storage technologies. However, as new tape technologies mature, scientific computing centers may switch to the tapes that best serve their needs.

2.1.3. Optical Disk and Tape

Recent advances in laser technology allow storage centers to use optical technology to archive data on both platters and linear media. Rather than recording data as changes in magnetic orientation of metal oxides, however, optical drives use microscopic pits in the media to store information. These pits are created and read by lasers in the drive. Since the creation of pits is not a reversible process, each bit on the medium can only be written once. Thus, optical disks and tapes are often referred to as WORM (write once read many) devices. Since optical tapes are particularly rare, WORM usually refers to optical disks.

Optical disks [41] are interchangeable media, unlike magnetic disks. Platters range from 5 inches to 12 inches in diameter, and usually contain data on both sides. While magnetic disks use concentric tracks to store data, optical disks have just a single spiral track on each surface. As Table 2-2 shows, optical disks have higher areal densities than magnetic disks. However, optical disks rotate slower than magnetic disks, so transfer rates are lower for optical disks. In addition, optical read/write heads are heavier and harder to move than magnetic read/write heads. As a result, average seek times for optical disks are approximately 100 ms — almost an order of magnitude slower than for magnetic disk. The combination of relatively low bandwidth and high media cost make optical disk unattractive for scientific computing centers, so few supercomputer centers use them in their storage hierarchies.

Optical tape is similar to longitudinal magnetic tape in most ways other than recording method. Optical tape, like optical disk, is a write-once medium that uses pits to encode data. The primary attractiveness of this medium is the amount of data that can be stored in a single cartridge. The CREO optical system [82] has

cartridges that can each hold a terabyte. However, both the media and the drive are expensive, as Table 2-2 notes. Additionally, each optical tape holds too much data — many supercomputer centers prefer to allocate each tape to a single user [39], and smaller tapes serve this purpose better. The LaserTape system [7] attempted to bridge this gap by providing drives and tapes with the same form factors as IBM 3490 drives and tapes. This technology allows supercomputer centers to increase storage capacity using existing tape robots. However, many centers such as NCAR [57,61] need the ability to rewrite tapes, eliminating optical tape from consideration.

2.1.4. Other Storage Technologies

Disks and tapes are the major storage technologies today. However, holographic storage [71] is a new technology that promises high bandwidth access to gigabytes of data contained in a cartridge a few centimeters on a side. Holographic storage systems use lasers to store data in three dimensions, as compared to tape and disk drives that only use the surface of the media to store data. A cube of holographic storage can provide storage on one of many parallel cross-sections of the solid, supplying approximately 1 GB of storage in a solid 4 cm by 4 cm by 0.5 cm. These media are interchangeable and might be used as part of a future system to store petabytes of data in a small space. [36] mentions a holographic storage system that will soon be available, though the storage media are read-only. There are no production read-write holographic storage systems, though prototypes exist. As a result, specific capacity and performance characteristics are difficult to predict.

2.1.5. Robotic Access to Storage Media

Massive storage is useless unless users can easily access it. Originally, computer operators loaded and unloaded media from drives. In the last decade, though, robotically managed storage has become widespread. Robots can load media faster and more accurately than humans, and allow the storage system to be fully automated. Robots are available for tapes and optical disks; those used for optical disks are often called *jukeboxes* (for their resemblance to the fixture in classic American diners) [95].

The tape robot most commonly used in supercomputer centers is the StorageTek Automated Cartridge System 4400 [48]. This robot stores 6,000 IBM 3490 tapes in a silo, using one of two arms to move tapes between their slots and one of several tape readers. The arms are very quick, and can pick a tape in less than 10 seconds. These robots are relatively expensive — approximately $250,000.

Exabyte makes a tape library system that holds 116 8mm tapes and four readers for approximately $40,000 [28]. While this system is considerably cheaper than the ACS 4400, 8mm tape drives transfer data more slowly than IBM 3490 drives. Long load and unload times and short tape lifetimes also make this system unsuitable for modern supercomputer centers. However, this system is sufficiently inexpensive that it might be used by smaller organizations, making tertiary storage ubiquitous rather than the province of only large data centers. Devices like this tape robot will allow workstations file systems to take advantage of tertiary storage, just as supercomputer file systems have.

2.2. File System Concepts

The storage devices mentioned in the previous section are only part of a complete storage system. These devices must be controlled by the file system — the software that manages data on secondary and tertiary storage devices. This section provides a brief overview of file system terminology and concepts using examples from the Berkeley Fast File System (FFS) [54,70] and the Log-Structured File System (LFS) [74]. Both of these file systems are designed for secondary storage. However, tertiary storage has been added to both systems. The Highlight file system [47] is an adaptation of LFS for tertiary storage, and Unitree [35,53] is a commercial integration of tertiary storage into Unix-type file systems. Both FFS and LFS are designed for workstations; while file systems for high-speed computers may use different techniques and data layouts,

the basic concepts underlying their design are similar. A more general discussion of file system internals can be found in [85].

This thesis is concerned with issues of data layout and storage allocation, not with higher-level file system functions such as name lookup and caching. Thus, this section concentrates on issues particular to the part of the file system corresponding to the *bitfile server* described in [13]. The bitfile server treats each files as simply a stream of bits with a numeric identifier. Translation of a user-friendly name to an identifier and caching of file blocks in memory is left to other parts of the file system.

A file system serves one basic purpose: the storage and retrieval of data from disks, tapes, and other forms of storage. To accomplish this, the file system must maintain its own data structures, *metadata*, as well as the data users entrust to it. Metadata includes information about the users' data such as internal file identifiers, timestamps, and permissions. It must also include sufficient information for the file system to translate a *logical address* — file identifier and offset within the file — to a *physical address* on the medium. Most disk-based file systems, including FFS and LFS, keep a list of disk blocks for each file, allowing the system to look up the physical block number for a specific block within a file. Tape-based file systems, on the other hand, may attempt to store an entire file sequentially on a single tape. Such a system would only need to keep a single tape identifier and offset from start of tape, since the offset of any block in the file could be easily found from that information.

The file system must maintain both internal consistency and data integrity. A file system is consistent if none of its data structures are corrupt. Consistency means that all blocks on disk are in a known state, and that each block is assigned to exactly one file or is unused. Integrity is a separate issue. A consistent file system may lose integrity if a data block is assigned to the incorrect file. While the conditions for consistency are met (all blocks belong to exactly one file), data blocks assigned to the wrong file can cause serious security breaches and loss of data. These two conditions are relatively easy to maintain during normal operation; however, remaining consistent and integral despite a system crash is more difficult and remains an issue in file system design.

2.2.1. Berkeley Fast File System

The Berkeley Fast File System (FFS) [54,67] is widely used, as it is a component of the BSD 4.3 operating system. This section gives a brief overview of its storage management and metadata structures.

FFS uses two sets of data structures to manage disk blocks: *inodes* and a free block map. An inode is a collection of metadata for a single of file. It contains creation, modification, and access timestamps as well as file size, ownership information and access permissions. In addition, the inode contains pointers to the physical addresses of the first few blocks in the file (the number depends on the particular implementation of FFS). The inode also contains pointers to *indirect blocks*, which are disk blocks containing lists of pointers to the remaining blocks in the file. Figure 2-4 shows the contents of an FFS inode and its associated indirect blocks.

FFS must also manage free disk space so it can quickly allocate unused blocks when needed. In FFS, the disk is divided into regions called *cylinder groups*, each of which contains one or more consecutive cylinders on disk. Each cylinder has its own set of inodes and its own free map. The free map is an array of bits corresponding to disk blocks in the cylinder group, allowing FFS to quickly find free blocks to assign to a file.

Consistency after a crash presents a problem for FFS. The file system must make two separate write operations to assign a free block to a file. The first write updates the inode or indirect block to point at the new data. The second write modifies the free map. These two writes are not sequential, so a crash between them will leave the file system inconsistent — a block will either be both free and allocated or neither free nor allocated, depending upon the order in which the operations are scheduled. FFS uses a file system check

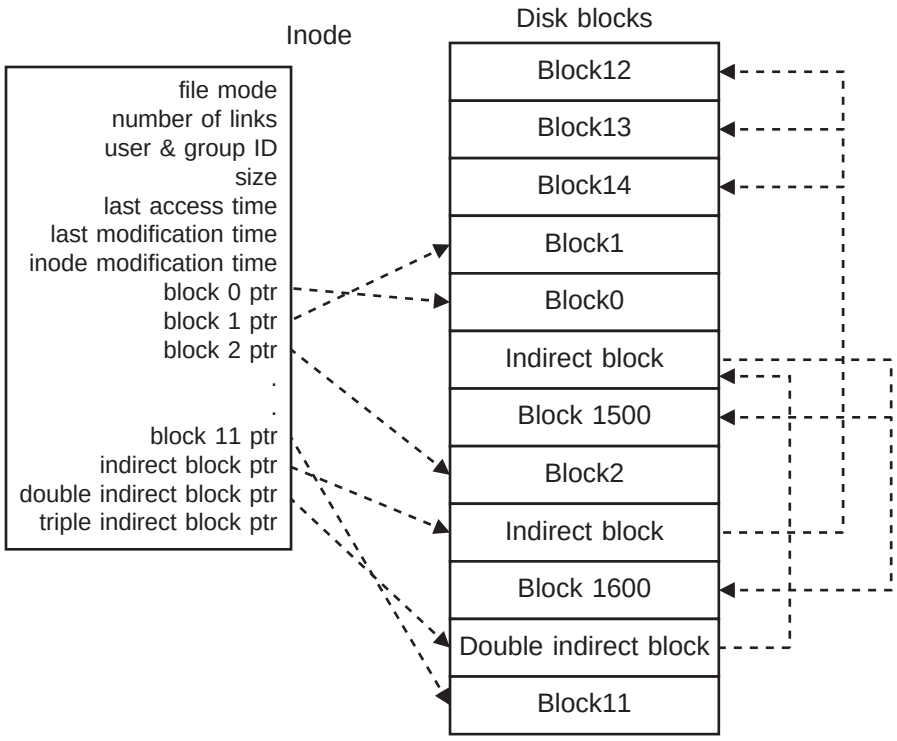

Figure 2-4. Inode and indirect blocks in the Berkeley Fast File System.

This diagram shows a single inode and its associated indirect blocks in the Berkeley Fast File System. Using this metadata, FFS can find any block in the file described by the inode. References to the first few blocks require no additional disk I/O to perform the translation of logical address to physical address. However, requests for the later blocks in large files may require two or three I/Os to follow the chain of indirect blocks.

program (`fsck`) after a crash to fix up the file system and restore consistency. While this is (barely) acceptable for a secondary file system, a full file system check on a tertiary storage system could take hours or longer.

2.2.2. Log-Structured File System

The Log-Structured File System (LFS) [74] was originally designed to run under the Sprite workstation operating system [66] and was later ported to BSD Unix [79]. It retains the same user-level semantics as standard Unix file systems, but uses different metadata structures and allocation mechanisms to improve performance and crash recovery.

LFS is based on the concept of *segments*. Segments are all the same size — typically 1 MB or longer — and are written in an atomic operation. Each segment may contain data and metadata. Rather than update in place as FFS does, LFS writes all modified blocks to the current segment, flushing the segment to disk when it has filled. Thus, creating a small file would result in the modification of the block containing the file's inode and the first block of the file. Both of these blocks would be written to the current segment, metadata first. While this arrangement works well to keep the file system consistent and integral, it creates a problem — finding the most recent inode written for a given file. Since the block containing the inode is rewritten each time it is modified, the disk may contain several copies (of varying ages) of the same inode. LFS

addresses this problem with a global map that translates file identifiers into pointers to inodes. This map must be updated separately from the metadata it points to. Since the map is merely a cache, however, the map may be quickly updated after a crash by scanning the most recent segments written.

The largest problem LFS must solve is "cleaning" the disk. Blocks in a segment become invalid as they are superceded by later writes. However, the file system is only willing to write data in empty segments. Thus, LFS must run a process to "clean" the disk by reading old segments, combining the live data from them, and writing the result as a new segment. The old segments are then marked as empty, ready to be filled with new data.

In contrast to FFS, LFS is a write-optimized file system. It assumes that writes to disk will dominate reads because of large caches. This is true for workstation file servers with large caches [2], but is not true for supercomputer workloads [60].

Neither FFS nor LFS is well-suited for scientific computation, as neither is optimized for large files. However, supercomputer and MPP file systems share basic design principles with these workstation file systems. They, like all file systems, must remain consistent and preserve data integrity. The RAMA file system design, which will be discussed in Chapter 4, uses techniques related to those used by FFS and LFS both for laying out data and maintaining consistency.

File systems for scientific computing must also provide high bandwidth access to large files and support the integration of tertiary storage. The next two sections discuss previous research efforts in these areas.

2.3. Mass Storage Systems

Tertiary storage devices such as magnetic tape have long been used to store scientific data. However, dramatic increases in computational power over the last two decades have produced ever-increasing quantities of data that must be stored. The demand for storage at scientific computing centers has primarily been satisfied by large tape systems using both nearline and offline storage. This great demand for storage has been satisfied in a largely *ad hoc* manner, however. Tertiary storage system designers have had little research to rely on, as there have been few studies of file migration and tertiary storage systems since [80] over a decade ago. This section discusses these repositories of scientific data and provides an overview of the research performed on tertiary storage systems and file migration.

2.3.1. Long-Term Reference Patterns and File Migration Algorithms

Mass storage systems for scientific computation first became common in the mid-1970's when computers began to generate more data than could be stored on a reasonable amount of secondary storage. By 1977, IBM had designed a tool to archive and retrieve data between disk and tertiary storage [16], though this paper only discussed the system's functionality and provided no analysis. [8] outlined several approaches to integrating mass storage systems into operating systems, but did not survey any specific systems. The first major study of such a system is detailed in [80] and [81]. In these papers, Smith analyzed the performance of the mass storage system at the Stanford Linear Accelerator System (SLAC) and proposed file migration algorithms to move files between disk and tape.

This study showed that the file reference distribution was skewed — most files were accessed infrequently, while a few files received many accesses. A file was considered referenced if it was accessed at least once on a given day; multiple accesses on the same day were counted as a single reference. The study found that the median number of accesses to a file was two, though the average number was 10.6. 41% of all files were referenced exactly twice, though fewer than 10% were referenced once. The remainder were referenced more than once, with 3.8% receiving more than 50 references. The study also showed that intervals between successive references to the same file were very short — more than 90% of all interreference intervals were

shorter than four days. The short interreference times and skewed file access pattern both make caching files from tertiary storage on disk an attractive proposition.

In [81], Smith went on to propose several file replacement algorithms to determine which files to keep on disk and which to move to tertiary storage. Each algorithm used age, file size, class, and time since last reference for every file to determine which files should be migrated to tertiary storage at the end of a day. While all of these parameters were available to each algorithm, most algorithms only used a subset of them. The best algorithms were those that used the entire reference history for the file system to estimate the time at which each file would be next used. While this method resulted in the highest hit rate, an algorithm that migrated files with the highest product of file size and time since last reference also performed acceptably. This algorithm is easier to calculate, and variations on it are still in use today. While this study provided a good foundation for future algorithm development, it did not consider environments where file transfer time is a major component of tertiary storage access time. As Chapter 3 will show, the large files used by scientists require more time to read than to access the first byte, thus affecting migration algorithms.

A study done on the University of Illinois computer center file access patterns in 1982 [49] confirmed the results reported in [80] and [81] and extended the migration algorithm analysis to include additional algorithms using a file's access history and file clustering. Using the full access history yielded little improvement over previous algorithms and, in some cases, actually performed worse. The latter technique, file clustering, migrates several related files at the same time, placing them on a single medium. While clustering files by user reduced the number of media accesses, it was marginally successful overall because of the number of superfluous files that were loaded. However, the file clustering analyzed in this paper performs less well on the large files common in supercomputing because the larger files exacerbate the problem of "false loads."

[31] detailed a third evaluation of file migration, but was the first to study a Unix file system. The study concluded that the Unix environment did result in some behavior different from that observed on other systems. Migrating files based solely on their size proved to be one of the best algorithms in this environment, contrary to the findings of previous studies. The paper made several recommendations for file migration in Unix based on the results of the analysis, including support for voluntary migration, special treatment for large files, and disk storage for all directories. For a Unix environment in the mid-1980's, these are good recommendations. However, they do not hold for scientific computing centers 10 years later, as this thesis will show.

The recent drop in the cost of robotic tape systems and increase in the amount of data that must be stored even at non-supercomputer sites has revived interest in file migration. [45] analyzed the behavior of the storage center at the National Center for Supercomputing Applications, yielding results similar to those reported in Chapter 3 of this thesis. The issue of file migration and long-term access patterns in Unix was revisited in [83], which found that file interreference intervals were still short, and that files stored on a Unix system were still relatively small — most less than 64 KB long. Simulations of the STP migration algorithm in this environment showed that it was effective, though the time component required different weightings for optimal performance on different file systems.

2.3.2. Existing Mass Storage Systems for Scientific Computing

The papers discussed in the previous section focus on quantitative analysis of existing systems and propose algorithms to solve the problems of managing a multi-level storage hierarchy. Much of the remaining literature on mass storage systems consists of "experience" papers in which operators of supercomputer data centers discuss their systems' behavior at a high level. These papers often include little specific data collection, instead focusing on the design and management of terabytes of data. Supercomputing centers discussed in practical experience papers include Lawrence Livermore National Laboratory (LLNL) [30,42], Los Alamos National Laboratory (LANL) [12,14,15], NASA Ames Research Center [39,72], and the

National Center for Atmospheric Research [32,65,86]. Data centers like these drove the development of the Mass Storage Systems Reference Model [13], providing a first step towards standardizing hardware and software interfaces for mass storage devices.

The storage designers at most data centers do not experiment with new file migration algorithms, as they lack the time to perform the full study necessary to justify changes in a new system. Instead, they make their "best guess" and implement migration policies that will work acceptably well. Once the system has been built, however, it is very difficult to implement modifications to migration policy even if analysis indicates they might be useful. Nevertheless, scientific computing centers have introduced some new concepts for mass storage. Unitree [53], a software package based on file migration software at LLNL, allows users to access their files without knowing whether they are stored on disk or tertiary storage. Similarly, the NASA Ames RASH project [39] allows transparent access to massive storage, and additionally clusters a single user's files together on tape. The RASH system also supports cut-through, allowing a user to access any bytes that have already been transferred from tape to disk while the rest of the file is still being moved. This method reduces the latency to read the first byte of data from tape, since a program need not wait for the entire file to be transferred.

While these studies introduced new techniques for designing mass storage systems, they did not design new algorithms. Instead, today's scientific data centers use file migration algorithms proposed by early studies that were designed for fewer and smaller files. To compensate, file migration algorithms are often modified *ad hoc* by system managers to tune performance for their particular system. This practice leads to better migration performance, but does not necessarily assist those designing their own mass storage systems. Many storage centers use algorithms based on the STP algorithms described in [81]. Rather than compute and sort the age-size product of every file, however, these systems often group files into bins of similar sizes and migrate the oldest files in each bin, approximating the space-time product as a migration criterion.

The integration of tertiary storage into a scientific computing environment has challenged system designers for many years. The research described in this section has guided the design of mass storage systems; however, additional research is needed to reflect the changes in storage requirements. Moreover, tertiary storage is but one piece of the data storage puzzle at most scientific data centers. The next section discusses previous research in parallel file systems, another important part of the storage hierarchy at modern centers for scientific computing.

2.4. Massively Parallel File Systems

As massively parallel computers have become more commonly used in scientific computing, they have added to the problem of data storage and management. Traditional file systems are controlled by a single processor and need no complex interprocessor synchronization and cooperation. Traditional supercomputers such as the Cray Y-MP achieve high file system bandwidth by using many disks in parallel under the control of a single CPU, as in a RAID (Redundant Array of Inexpensive Disks) [10,50]. This approach has limited scalability, as a single CPU cannot control an infinite number of disks. System designers may get around the problem by using multiple disk controller CPUs; as long as requests for a file come only from a single processor, issues of coherency and synchronization are largely avoided. Distributed file systems such as Sprite [66], in which several computers on a network share a single file system, require more complex protocols. However, files in such systems are rarely shared by two CPUs at the same time [85]. The reuse of a file by different computers at different times adds some complexity, but the file system CPU usually only needs to field requests for a specific file from one computer at a time.

Massively parallel processors present two major problems to file system designers. First, the individual processors in an MPP cooperate on a single problem and access different parts of a single file at the same time. No longer can a file system assume that the majority of accesses to a file at any time will come from a single CPU. This issue affects the design of caching and coherency schemes for parallel file systems.

The second problem that MPPs present is scalability. A uniprocessor's request rate scales with the speed of a single processor — as technology allows the processor making the requests to run faster and make more requests, it also allows the (single) CPU running the file system to handle more requests. A distributed system can provide a unified view of multiple file systems each with its own CPU to provide additional performance as hosts are added to the network. However, an MPP requires scalable performance when accessing a single file. Doubling the number of nodes in an MPP may require the data and request bandwidth for a single file to double without providing a faster CPU to run the file system. Thus, MPP file systems must allow multiple CPUs to run a single file system to guarantee scalability. Adding additional disks to provide higher bandwidth is relatively straightforward; controlling them to provide higher file system performance is a problem this thesis will address.

This section will discuss previous work done on parallel file systems. This work falls into two broad categories: general research on parallel file systems and specific file system designs and analyses. Some research, such as discussions of access patterns and methods, is applicable to many different parallel file systems, since it discusses an application's view of the file system. The first part of this section will discuss such work. The remainder of the section covers analyses of parallel file system simulations and implementations.

2.4.1. File System-Independent Parallel I/O Improvements

Much research on parallel file systems has focused on high-level concerns about parallel file systems. These papers are not concerned with data distribution on disk; rather, they discuss how parallel applications use a file system and provide guidelines for future designers. Several papers on specific file systems also address these issues; such papers will be discussed later.

[22] provides a good overview of the types of file accesses that parallel applications make of file systems. It divides files into two types: those transferred sequentially, and those accessed randomly. These access patterns are shown in Figure 2-5. As the diagram shows, a parallel file system must support interleaved access and sequential access. In addition, a parallel file system must support random access by many processors to a single file, an access pattern called general direct access in [22]. The paper only suggests methods for building such file systems, however, and does not discuss any real implementations.

The issue of reordering data between the I/O system and the application is discussed in [23]. This paper suggests that the application and the file system should each pick their "preferred" layout, and allow the file system software to reorganize the data in two phases. During the first phase, large sequential chunks of data are read from the disks to the nodes. This strategy uses the disks effectively, but the data does not necessarily reach the processor that needs it. The second phase accomplishes this task, using the high-bandwidth interconnection to redistribute the data to the appropriate nodes. Two-phase I/O was tested on both the Touchstone Delta [5] and the nCUBE-2 [64], providing speedups between 1.5 and well over 100, the latter occurring when a row-major matrix was read in column-major order. These speedups depend on all of the processors in the MPP cooperating to issue their file requests, since the file system must know all of the data to be transferred before it can start. This method is thus promising for parallel applications that can support such a model, but it provides less help to programs that do not have regular access patterns.

2.4.2. Bridge

The Bridge file system [24,25] integrates applications tightly into the file system to gain better performance. Data is striped across many disks, each of which has its own local file system. Bridge provides a unified view of these local file system to applications and adds support for parallel operations such as opens, reads, and writes. Individual applications may, however, learn the low-level layout of a Bridge file on disk and optimize their actions accordingly.

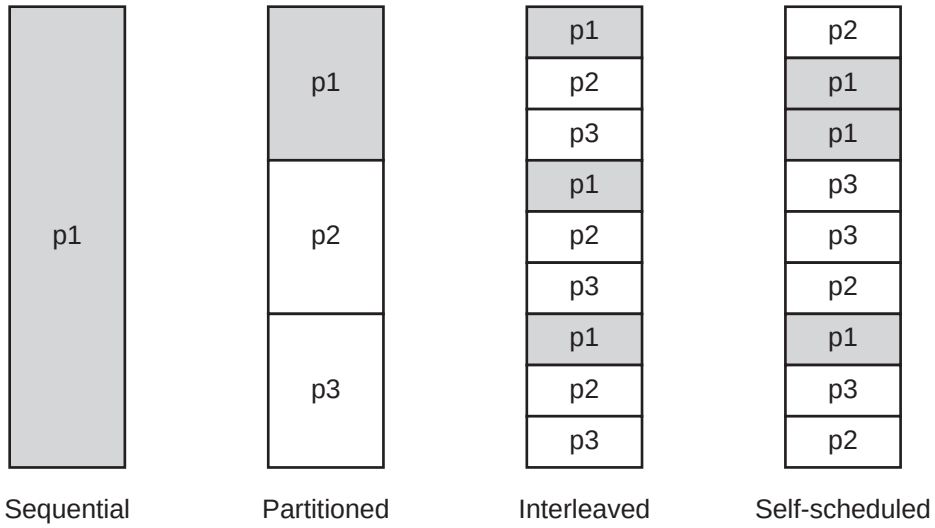

Figure 2-5. Sequential file access patterns in parallel programs.

[22] describes several access patterns with which parallel programs access a single file. The access patterns shown in the diagram above apply to programs that step through a file in a regular fashion, with the various patterns corresponding to the division of work in the program. In the partitioned and sequential cases, the program assigns large chunks of the file to a single processor. The interleaved case is similar, but the chunks of the file allocated to each processor are not sequential, though the division is regular. A self-scheduled program divides the file into chunks and allows the processors to "compete" for each piece. The file is read sequentially in time, but the processor that reads a given chunk is determined by which processor is free to do the work.

Bridge performed well for straightforward operations such as file copies. Adding additional processors and disks gave a linear speedup on sequential reads and writes. The authors also designed other tools to test their file system; these similarly achieved speedup with more processors and disks. However, the tools all relied on low-level knowledge of the file system. While this approach provides good performance and parallelism, it is not easily portable and requires a great deal of programmer effort.

2.4.3. Concurrent File System

Intel's Concurrent File System (CFS) runs on several massively parallel processors — the iPSC/2 hypercube [68] and, after some evolutionary changes, the Touchstone Delta mesh [6,5]. Both machines have separate compute and I/O nodes, restricting the file system software to only those nodes with disks. CFS attempts to preserve the Unix file interface; however, several primitives for parallel file access are included.

[68] found that I/O scaled as nodes with disks (*disked* nodes) were added. The example in the paper worked best with one I/O node for each compute node — fewer compute nodes could not drive all of the I/O nodes to full bandwidth, and fewer disked nodes could not supply data rapidly enough. However, the test only included up to eight I/O nodes and 16 compute nodes. The experiments run on the Touchstone Delta were more complete. They showed that file system bandwidth was independent of the node on which the data was physically stored. While certain access patterns performed worse than others, this drop was due to poor distribution of the file to disk, not the location of the disked node within the interconnection network. The file system rarely attained the maximum possible hardware throughputs, largely due to software overheads and poor file system cache management. Because of the poor file system performance, most programs that use the file system on the Touchstone Delta are I/O bound.

2.4.4. Vesta

Vesta [18,19] is an experimental file system that runs on the IBM SP-1 [43], a parallel processor composed of workstations connected by a high-speed interconnect. Like the Bridge file system, Vesta stripes data across the disked nodes in the system using the existing workstation file system on each node, and provides a single parallel file system view to applications. Rather than describe a file as a linear sequence of bytes, however, Vesta promotes a two-dimensional file structure to correspond to many two-dimensional array layouts in software. Application software must specify the physical layout when the file is created. Additionally, each application may create a *view* of the file, assigning a logical order to the data in the file. The view also allows a file to partitioned into disjoint units, allowing each processor to access its own view without the need for synchronization with other processors.

Two-dimensional physical layout and views allow Vesta to optimize multiprocessor applications' access to parallel files. Vesta also supports a disk equivalent of scatter-gather — it collects all of the requests for parts of a parallel file, melds them into the minimum possible number of disk requests, and then performs the requests. Rather than send a message for each individual file request, Vesta aggregates the data and sends fewer but larger messages. These techniques allow Vesta's performance on sorts and other MPP applications to show good speedups as the number of processors and disks increases.

As with Bridge, however, the programmer must expend a good deal of effort to insure that the file is laid out and partitioned correctly. [19] reports that reactions to the file access models supported by Vesta ranged from enthusiastic to very reluctant. While Vesta performs well, it is not clear that application programmers want the trouble necessary to get such performance.

2.4.5. CM-5 File System (*sfs*)

The Thinking Machines CM-5 [88] MPP uses a file system called *sfs* [52], which is based on the Unix Fast File System [54]. The CM-5 has separate compute and I/O nodes; while the disked nodes use the same SPARC processor as the compute nodes, they require additional hardware to run the attached disks. Each Disk Storage Node (DSN) has eight 1.2 GB drives, two on each of four channels. These DSN units are packaged in groups of three, so a CM-5 will typically have a multiple of 24 disks in its file system.

This file system, like many others, stripes data across as many as 100 disks or more. As a result, peak performance for large file systems is only reached with very large requests. A file system with 118 disks was able to transfer at 175 MB/s, but this transfer rate was only reached when over 200 MB were transferred. Nonetheless, the file system was able to maintain per-disk transfer rates of 1 MB/s for writes and 1.5 MB/s for reads. Thus, *sfs* provides good performance for large transfers, but left unaddressed the issue of many small requests coming from hundreds of processors.

2.4.6. Uniprocessor File Systems Used by MPPs

Many MPPs do not actually use a truly parallel file system. Instead, they use a high-speed uniprocessor file system attached to the MPP by a high bandwidth connection. The MPP then accesses the foreign file system through an interface host. This arrangement allows an MPP to use existing file system software and hardware, and minimizes the software complexity for the file system. However, this approach is not easily scalable, and cannot easily accommodate many independent requests from the hundreds of processors in the MPP.

Variations on this approach are used in the Thinking Machines CM-2 [87] and the initial release of the Cray T3D [21,20]. The CM-2 file system is actually run by a workstation which coordinates the bitwise striping of data across tens of disks. Since the CM-2 is a SIMD machine, the number of requests to the file system does not scale with the number of processors, though the size of a request may. Thus, the single processor

of the workstation can keep up with a larger CM-2, and additional bandwidth can be added by attaching more disks.

The CM-2 approach works well for SIMD parallel processors, but breaks down for MIMD machines where each processor makes its own requests. The initial release of the Cray T3D operating system uses a Cray Y-MP class machine to run the file system. All requests are satisfied by this host, and all data transfers are coordinated by it. The Y-MP uses disk arrays for its file system, providing support for high bandwidth and high request rate. However, this scheme is not scalable. The Y-MP does not become faster as more nodes are added to the T3D, creating a bottleneck for file access. Cray has recognized this problem, and plans to design a file system that runs on the nodes of the MPP rather than on a separate machine [20].

2.4.7. Other Parallel File Systems

The file systems previously mentioned in this section are only some of the parallel file systems proposed or developed. While other parallel file systems are less-studied, they may provide some useful design concepts.

The hashed-index file system (HIFS) [3] uses hashing to distribute file blocks to disked nodes in a scheme similar to that described in Chapter 4. HIFS thus allows file access without any centralized indexing, allowing any processor to locate data using just a hash function. This system is proposed for parallel databases where the application typically reads only a few kilobytes from any location in the file; thus, HIFS distributes each block to a different node and does not cluster sequential blocks. The projected performance of such a system scales with more disks and more processors, with the speedup approaching linear. However, maximum speedup depends on having a sufficiently large number of simultaneous requests. This performance is similar to that described in Chapter 6.

[92] describes a parallel file system for the Hypercube Multiprocessor. The file system supports cooperative access to a single file by multiple CPUs. However, files are stored on a single disk and not distributed among several disks. Performance is thus limited by the bandwidth available from a single disk. [29] proposes an alternate file system for the Hypercube that distributes data from a single file to multiple disks. The design uses a separate network to handle I/O traffic, alleviating the load on the relatively slow links between processors.

The TickerTAIP parallel disk system [9] is actually a block server, not a true file system. However, it could be extended to provide bitfile service as described in [13]. The disks in TickerTAIP are controlled by multiple cooperating CPUs, eliminating the traditional RAID bottleneck and allowing the system to scale to hundreds of disks. Parity management is also distributed, providing a potential model for distributing parity in a parallel file system. TickerTAIP is noteworthy both for its design innovations and for its demonstration that interconnection link speeds of less than 2 MB/s were sufficient to provide 12 MB/s of data service.

2.5. Conclusions

This chapter has presented the background material on which the rest of this thesis will build. Storage management is an important part of the scientific computing environment, yet current tertiary storage systems and parallel file systems need improvement. Software is only part of the picture, however. Storage systems may be built from a wide variety of devices from magnetic disks to robotic tape systems to optical disk. Software and hardware must work well together to provide a high-speed storage system with a multi-terabyte capacity.

A modern mass storage system designer can choose from several types of storage devices, each with advantages and drawbacks. The primary tradeoff a designer must make is between cost and performance. Devices such as memory have low access latency and high bandwidth but cost $50,000 per gigabyte. Robotic systems for magnetic tape and optical disk, on the other hand, can cost as little as $100 per gigabyte, but have

bandwidths and access latencies as much as nine orders of magnitude slower. Magnetic disks, intermediate between these two extremes in both cost and performance, are used both as caches for tertiary storage and temporary storage for data that does not fit in limited semiconductor memory. Design decisions are complicated by the cost and performance variations within in each class — for example, different magnetic tape technologies have transfer rates ranging from 0.5 to 15 megabytes per second.

Mass storage systems for scientific computing are typically built from magnetic disks and robotically-managed magnetic tape. Most of the data at a supercomputing center is stored on magnetic tape, while the most recently used data is cached on magnetic disk. This approach should provide a system with the speed of magnetic disk and the cost of magnetic tape. However, maintaining this illusion requires finely-tuned file migration algorithms to decide when to move data between tape and disk. Much of the work done on file migration algorithms only considered the time necessary to read the first byte of a file; files were small so it took relatively little additional time to read the remainder of the file from a tape. Simple variations on these early algorithms have been used in mass storage systems for over a decade. However, the implementers of these storage systems can experiment with neither migration algorithms nor device configuration — once a system is built, it is very difficult to convince users to survive without the system while it is being reconfigured. Thus, new analysis is necessary to give system designers a more current picture of the behavior of mass storage systems.

Parallel file systems, also components in a storage system supporting scientific computation, are less mature than tertiary storage system. While the supercomputing community has agreed on some standards for mass storage systems [13], they have not agreed on the best model for file systems on massively parallel processors. Some systems simply use uniprocessor file systems attached via high-bandwidth network, while others stripe data across disks attached directly to the parallel processor. Both models share several problems, though. Multiprocessor file systems present a bottleneck in moving data between a parallel application and disk. Their performance can be improved, but only at the expense of programmer effort, often using knowledge of the underlying hardware and limiting an application's portability. Additionally, the special structures used in most parallel file systems make them difficult to integrate into an environment with workstations and tertiary storage; the current solution is to manually copy files between the MPP and the rest of the world. The parallel file system developed later in the thesis addresses this problem.

The remainder of this thesis discusses solutions to several problems brought up in this chapter. Chapter 3 discusses a recent study of a typical modern mass storage system, comparing it with earlier studies of such systems and highlighting recent changes. It then discusses how these changes might affect the design of new mass storage systems. Next, Chapters 4 through 7 present a parallel file system that addresses several of the issues raised in this chapter — integration into tertiary storage, ease of use, and performance.

3 File Migration

The first problem this thesis addresses is that of providing the terabytes of storage that scientists typically need to store their data. The dramatic gains in the speed of supercomputers used in science have encouraged the processing of ever larger amounts of data; however, storing this data on magnetic disk is not cost-effective. A typical data center stores more than 20 terabytes of data; this storage would require 20,000 1 GB disks costing more than 20 million dollars and 100 standard 6-foot high racks to hold them. Power consumption and maintenance would also be a problem, as standard disks have a mean time between failure (MTBF) of around 200,000 hours. Even using this liberal figure, such a system would experience, on average, about 2.5 disk failures per day.

Instead, most data centers with large data sets use tertiary storage devices such as tapes and optical disks to store much of their data. These devices provide a lower cost per megabyte of storage — a tape cartridge can cost as little as $5/GB as compared to nearly $1000/GB for disk — but they have longer access times than magnetic disk. By studying the tradeoffs between cheaper and slower tertiary storage and more expensive and faster disk storage, response time can be improved without increasing storage costs.

This chapter is a case study of one such site — the National Center for Atmospheric Research (NCAR), which, like many other supercomputer centers, deals with large amounts of data that can never be deleted. Data grows at the rate of several terabytes per year [90]. The cost of storing this data on shelved magnetic tape is relatively low, as tape cartridges cost less than $100/GB (see Section 2.1). However, storing even 1% of the total data in magnetic disk would be expensive, requiring hundreds of gigabytes of Cray disk storage. Instead, NCAR uses tertiary storage devices, primarily IBM 3480 tapes, to store their data.

This chapter analyzes file migration behavior in the NCAR system described in [1] and [86]. The first section of the chapter describes the NCAR system, and notes the features that might affect tertiary storage usage. This is followed by a discussion of the trace-gathering and analysis methods.

The main part of the chapter is a two-part analysis of the gathered trace data — analyzing the overall usage patterns for the entire mass storage system (MSS), and studying the behavior of individual files. The first part of the analysis includes system behavior over the course of a day, a week, and longer periods. It characterizes user behavior with respect to the entire MSS, showing at what rate data and files are read and written. We discuss other characteristics of the mass store at NCAR, such as request latency and interrequest distribution. The second part of the analysis provides insight for designing migration algorithms, as it focuses on how individual files are treated. This part of the analysis will discuss file size distribution and individual file reference patterns.

The chapter concludes with a discussion of the implications of the analysis on migration algorithms, and suggests some directions for future research. One of the major findings reported in this chapter is that tertiary storage must be well-integrated with secondary storage. Additionally, the tertiary storage system must

be able to prefetch files without knowing how they will be used once they are on disk. This problem is further complicated by the increasing use of massively parallel processors (MPPs) in scientific computing centers. It is this combination of requirements that demonstrated the need for RAMA, the file system described in the remainder of the thesis. RAMA addresses these issues by providing high-performance file access to MPPs without requiring application placement hints. RAMA also integrates tertiary storage well, keeping the same namespace for tertiary and secondary storage and allowing individual file blocks rather than entire files to migrate.

3.1. NCAR System Configuration

This section describes the system on which the file migration traces were gathered. The emphasis in this section is on those parts of the NCAR system that are relevant to the study in this chapter; however, the rest of the environment will be briefly described, as the mass storage system is shared by all of the systems at NCAR, possibly affecting the performance of the mass storage system. The NCAR system's storage devices are similar to those at other mass storage systems, such as those at NASA Ames Research Center and Lawrence Livermore Labs. Additionally, the researchers at many sites use both supercomputers (typically Crays, though MPPs are becoming more common) and workstations to do their research, which involves modeling complex processes and analyzing the results of these models. It thus seems that NCAR is typical of MSS centers.

3.1.1. Hardware Configuration

The CPU in the study was a Cray Y-MP 8/864 (`shavano.ucar.edu`), with 8 CPUs and 64 MWords[1] of main memory. Each CPU has a 6 ns cycle time. `Shavano`, like other Cray Y-MPs, has three 100 MB/sec connections to its local disks and two 1 GB/sec connections to a solid state disk (SSD). There are approximately 56 GB of disks attached directly to the Cray. 47 GB of this space is reserved for application scratch space and files over a few days old are purged from it regularly.

The mass storage system (MSS) at NCAR is composed of an IBM 3090 processor — used as a bitfile[2] server — with 100 GB of online disk on IBM 3380s, a StorageTek Automated Cartridge System 4400 with 6000 200 MB IBM 3480-style cartridges, and approximately 25 TB of data in shelved tape. The MSS tries to keep all files under 30 MB on the 3090 disks, and immediately sends all files larger than 30 MB to tape. Usually, the tapes written are those in the cartridge silo. Files on the MSS are limited to 200 MB in length, since a file cannot span multiple tapes.[3] While the Cray supports much larger files on its local disks, they must be broken up before they can be written to the MSS.

The MSS at NCAR is shared by the entire NCAR computing environment, which includes the Cray Y-MP, an IBM 3090 which runs the MSS, several VAXen, and many workstations. Figure 3-1 shows the network connections between the various machines at NCAR. The disks and tape drives attached to the MSS processor have direct connections to the Crays via the Local Data Network (LDN), providing a high-speed data path. All machines connected to the MSS (including the Crays) are connected to the 3090 by a custom hyperchannel-based network called the MASnet. Data going out over the MASnet must pass through the 3090's main memory, so it is a slower path than the direct connection the Crays have. The few workstations with connections act as gateways to the networks which connect to the rest of the workstations at NCAR. These gateways are also the fileservers for the local networks. Many of these smaller machines have their own local lower-speed disks. Approximately 5.5 GB of these disks are mounted by the Cray via NFS (Net-

1. Each Cray word is 8 bytes long.
2. A bitfile is a stream of bits stored by the file system. It is the same as a plain file in UNIX.
3. This is a software limitation, and will be fixed soon.

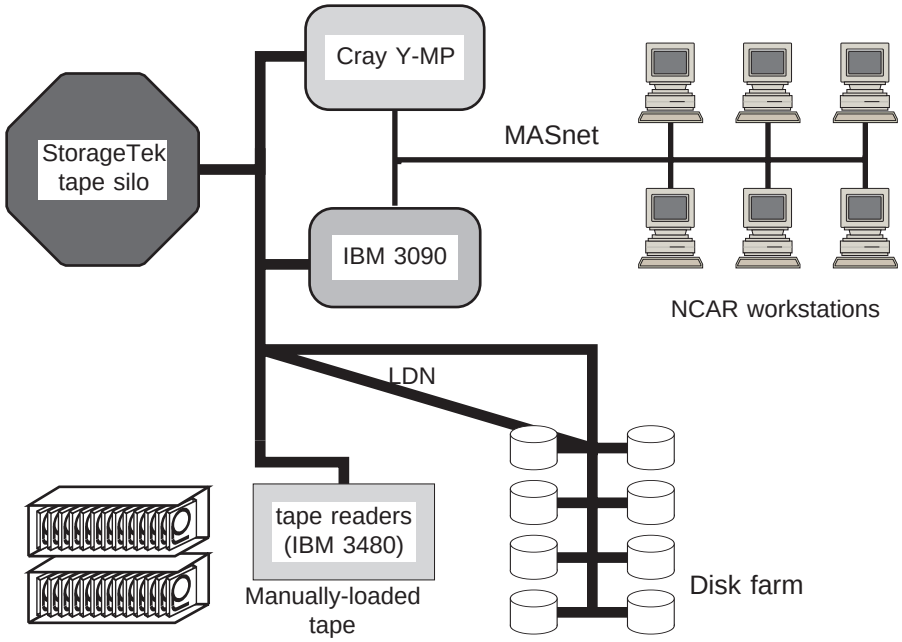

Figure 3-1. The NCAR network.

This figure details the connections between supercomputers, workstations, and the mass storage system (MSS) at NCAR. The MSS consists of massive disk storage, robotically-managed tape storage, and shelved tapes mounted by human operators.

work File System). According to the monthly report published by the NCAR systems group [90], `shavano` puts more data on the network than any other node, but several other nodes receive more data. In particular, several of the Sun workstations receive comparable amounts of data. It is likely that these workstations, which are the gateways to internal networks of desktop workstations, are receiving a large amount of visualization traffic.

3.1.2. System Software

NCAR scientists primarily use the Cray Y-MP for climate simulations— both the extensive number crunching necessary to generate the data, and the less computationally-intensive processing to visualize it. The Cray has two primary modes of operation: interactive or batch. In interactive mode, programs are short and run as the user requests them. In batch mode, however, jobs are queued up and run when space and CPU time are available. There is no explicit switch between operating modes, but short interactive jobs typically have higher priority. These interactive jobs fall into two categories: short commands run on the Cray and debugging runs of modeling programs. Since such jobs are submitted only when there are scientists around to submit them, interactive mode dominates during the day. Batch mode takes over on nights, weekends, and holidays when there are few people making interactive requests. The MSS request patterns reflect these two different uses of the CPU, as will be shown below.

The architecture of the software which runs the MSS is based on the Mass Storage Systems Reference Model [13]. It consists of software on the mass storage control processor (MSCP), which is the IBM 3090, and one or more bitfile mover processes on the Cray. Users on the Cray make explicit requests (via the UNICOS commands `lread` and `lwrite`) to read or write the MSS. These commands send messages to the MSCP, which locates the file and arranges for any necessary media mounts. The MSCP then configures

the devices to transfer directly to the Cray. For disk and tape silo requests, these mounts are handled without operator intervention; however, an operator must intervene to mount any non-silo tapes which are requested. After the data is ready to be transferred, the MSCP sends a message to a bitfile mover process which manages the actual data movement. When transfer is complete, the bitfile mover returns a completion status to the user.

3.1.3. Applications

The Cray at NCAR runs two types of jobs: interactive and batch. Interactive jobs finish quickly and require a short turnaround time. Batch jobs, on the other hand, may require hours of CPU time but have no specific response time requirements.

A typical climate simulation, such as the Community Climate Model [91], runs for 2 minutes of Cray Y-MP CPU time per simulated day and writes 50 MB of data during that time. Thus, a run of the model that simulated 10 years of climate would require 5 full days of computer time, and write 182 GB of data, averaging 600 KB of data per CPU second. This is an example of a batch job; a researcher would submit the job, doing other work until it finished. These jobs use a large amount of temporary disk storage as well as CPU time, because Cray programs do not write data directly to tape. The Y-MP at NCAR is configured with small, 300 MB user partitions. Each user is allocated a few megabytes on one partition, which would be insufficient for storing the output of even one run of a climate model. Thus, the initial input to a climate model must come from the MSS, and any results must go back to the MSS. If the results are needed later, they must be retrieved from the MSS.

Interactive jobs, such as a “movie” of the results of a climate simulation or a global average temperature over the course of a simulation, have much more stringent response time requirements. Typically, a user will initiate a command and expect a response quickly. According to [89], an interactive request must be satisfied in just a few seconds, or interactive behavior is lost. Nevertheless, the average response time to satisfy MSS requests is over 60 seconds; possible solutions to this problem will be discussed later.

3.2. Tracing Methods

3.2.1. Trace Collection

The data used in this study was gathered from system logs generated by the mass storage controller process and the bitfile mover processes. Approximately 50 MB of data were written to these logs per month. The system managers at NCAR use the data to plan future equipment acquisitions and improve performance on the current system. The logs also serve as proof that a requested transaction took place, as system managers occasionally use them to refute users who claim their files were written to the MSS and then disappeared.

The system log, as written by the mass storage management processes, contains a wealth of information. Much of it is either redundant or unnecessary for analyzing file migration behavior. Information such as project number and user name are not needed for migration studies, since the user identifier is also reported. The trace logs are designed to be easily human-readable, so fields are always identified and dates and times are in human-readable form. In addition, each MSS request is assigned a sequence number, since there are several records in the system log which correspond to the same I/O. While this method makes generating the trace logs easier, it is very wasteful of space. However, the sequence numbers can be used to generate much more compact traces, with only one entry per file access. By processing the traces to remove redundant information and transforming the rest of the information into a form more appropriate for machine processing, the traces were cut from 50 MB per month to 10-11 MB per month. They could not be reduced further because file names are long and cannot be compressed without losing vital information such as file placement in the directory structure.

3.2.2. Trace Format

Once the system logs were copied to a local host, they were processed into a trace in a format that is easy for a trace simulator or analysis program to read. The traces were kept in ASCII text so they would be easy to read on different machines with different byte orderings. A list of the fields in the trace is in Table 3-1.

Field	Meaning
source	Device the data came from
destination	Device the data is going to
flags	Read/write, error information, compression information
start time	time in seconds since the previous start time
startup latency	time in seconds to start the transfer
transfer time	time in milliseconds to transfer the data
file size	file size in bytes
MSS file name	file name on the MSS
local file name	file name on the computer
user ID	user who made the request

Table 3-1. Information in a single trace record.

This is the information included in a single trace record. The sequence numbers from the original trace logs were not kept since their only purpose was to allow multiple entries for a single file access to be correlated. Other information such as project number (used for billing) and other accounting information was also discarded. The original traces are kept in the MSS at NCAR, and can be reread if more information from them is necessary.

Two consecutive records in the trace have little information in common except their time stamps, which are presumably close together. Even so, the traces are compressed by recording times as differences from some previous time, as suggested in [76]. The start time for an MSS request is recorded as the elapsed time since the start time of the previous request, while the latency until the first byte is transferred (the startup latency) and the transfer time are recorded as durations. Recording timestamps as differences reduced the trace size by 30% compared to keeping the absolute times in the trace. Start time and startup latency are measured in seconds, while transfer time is measured in milliseconds. These were the precisions available from the original system logs. The only other commonality between consecutive requests might be the requesting user, so there is a bit in the flag field which indicates that the request was made by the same user who made the previous request. Directories, too, might be common between consecutive requests, but they would be harder to match. Future versions of the trace format may allow for full or partial paths to be obtained from previous records.

3.3. Observations

The traces for this study were collected over a period of 24 months, from October, 1990 through September, 1992. Traces were available from the time the MSS came on-line in June, 1990, but the MSS was very lightly used for the first few months as it was put into full production use. By October 1990, the request rate to the MSS reached a level comparable with its long-term behavior from that point forward. Since the steady-state behavior of the MSS was the target of the study, the traces during the startup usage period were omitted. To study long-term trends in MSS access, however, the tracing period started as soon as the system

reached steady-state, rather than waiting for a few months after that point. This insured the longest possible window to study long-term MSS behavior.

3.3.1. Trace Statistics

Overall statistics for the trace period are shown in Table 3-2. The traces actually include 3,688,817 references, but 175,633 (4.76%) had errors, the frequency of which is shown in Table 3-3. The most common error, fully 85% of all access errors recorded in the trace and 4.06% of all references in the trace, was the non-existence of a requested file. In such cases, it was impossible to include the reference in the analysis, since the file never existed, and thus had no source or destination location. It might have been possible to include references that encountered other errors, such as media errors and premature termination. However, such errors made up only 0.7% of all of the references, and many of these errors, such as migrating too large a file and not having the proper permissions, also resulted in no actual data movement.

	Reads	**Writes**	**Total**
References	2336747. (66%)	1179047. (33%)	3515794. (100%)
Disk	1419280. (60%)	927722. (39%)	2347002. (66%)
Tape (silo)	480545. (66%)	239162. (33%)	719707. (20%)
Tape (manual)	436922. (97%)	12163. (2%)	449085. (12%)
GB transferred	63926.2 (73%)	23389.9 (27%)	87316.2 (100%)
Disk	5080.4 (58%)	3727.9 (42%)	8808.3 (10%)
Tape (silo)	38256.6 (67%)	19081.4 (33%)	57338.1 (66%)
Tape (manual)	20589.2 (97%)	580.6 (3%)	21169.8 (24%)
Avg. file size (MB)	27.36	19.84	24.84
Disk	3.58	4.02	3.75
Tape (silo)	79.61	79.78	79.67
Tape (manual)	47.12	47.74	47.14
Secs to first byte	98.1	38.6	78.18
Disk	32.47	25.39	29.67
Tape (silo)	115.14	81.86	104.08
Tape (manual)	292.58	203.84	290.18

Table 3-2. Overall NCAR trace statistics.

The trace covers the period from October, 1990 through September, 1992. The percentages listed under “Reads” and “Writes” are ratios to the value in the “Total” column of that row. The percentages listed under “Total” are percentages relative to the top value in the column.

Table 3-4 contains data about the massive store to which the references were made. This table only includes statistics for files which were referenced during the trace period. Since data on the actual contents of the MSS was unavailable, the study assumed that only files actually referenced during the trace period existed on the mass store. This is a valid simplification, as there are only three kinds of files that are never explicitly read or written—large temporary files used by Cray applications, small files that fit into the 1 MB allocated for each user’s home directory, and system files such as binaries. The first category, temporary files, would

Error Type	Number of Errors (Percent)
File not found	149925 (85.3%)
File creation failed	5014 (2.8%)
User aborted the access	2982 (1.7%)
File too large to migrate	2878 (1.6%)
User not allowed to access file	1286 (0.7%)
Media error	873 (0.5%)
File operation attempted on a directory	858 (0.5%)
Other errors	11817 (6.7%)
Total	175633 (100%)

Table 3-3. NCAR MSS access errors.

This table lists the major error types listed in the NCAR traces. Only the most common errors, accounting for 93.3% of the total errors in the traces, are listed separately here. “File not found” errors occurred on 4.06% of all accesses to the MSS, and only 0.70% of all accesses encountered any other kind of error.

be actively used for their entire lifetime, and discarded when no longer in use, never providing a chance to move them to long-term storage. Small user files, such as `.cshrc`, would never be migrated since they are used too often. Even if they were migrated, they would only add approximately 4 GB of space to the MSS, assuming each of the 4,000 users filled their entire permanent allocation. System files, likewise, would probably be used often enough so they would not be evicted from disk. Additionally, most system files are read-only, eliminating the need to write any data to the MSS.

Number of files	902772
Average file size	25 MB
Number of directories	143245
Largest directory	24926 files
Maximum directory depth	12
Total data in MSS	23 TB

Table 3-4. NCAR MSS overall statistics.

For the MSS to be large enough to satisfy all of the traced accesses, it would have to be at least as large as indicated in the table. This is a minimum size, however, since we only traced accesses from the Cray Y-MP at NCAR. Accesses from other computers, such as the workstation network, might increase these requirements.

3.3.2. Latency to First Byte

Figure 3-2 shows the total latency from when a request is made to the MSS until the data transfer actually starts. This time is composed of several elements—queueing time on the Cray, queueing time on the MSS, media mounting time, and seek time. For the disk, media mounting time and seek time are very short, usually well under a second. While median access time for the disk was 4 seconds, the distribution has a long

tail due to queueing at individual disks. Each disk has a relatively low bandwidth, so a large file takes several seconds to satisfy. Any requests for this disk that arrive in the meantime must wait for the long request to finish, generating the long delays in the tail of the disk latency curve.

Delays were caused by queueing in several places in the system—the Cray, the MSS CPU, the network from disk to Cray, and data transfer to or from the device itself. Of these, the only delays that differ between disk, tape silo, and shelved tape are the latencies due to the device itself— queueing at the device, transfer delays and seek delays. The disks do not transfer data much faster than the tape drives, so queueing delays for them may be representative of the time spent waiting for a tape drive to become available. On the other hand, the long latency to the first byte of data on a tape might cause tape queues to be longer because the tape readers are tied up even while the tape in them is rewinding. Since it takes longer to complete an access, the queue for the device could well be longer.

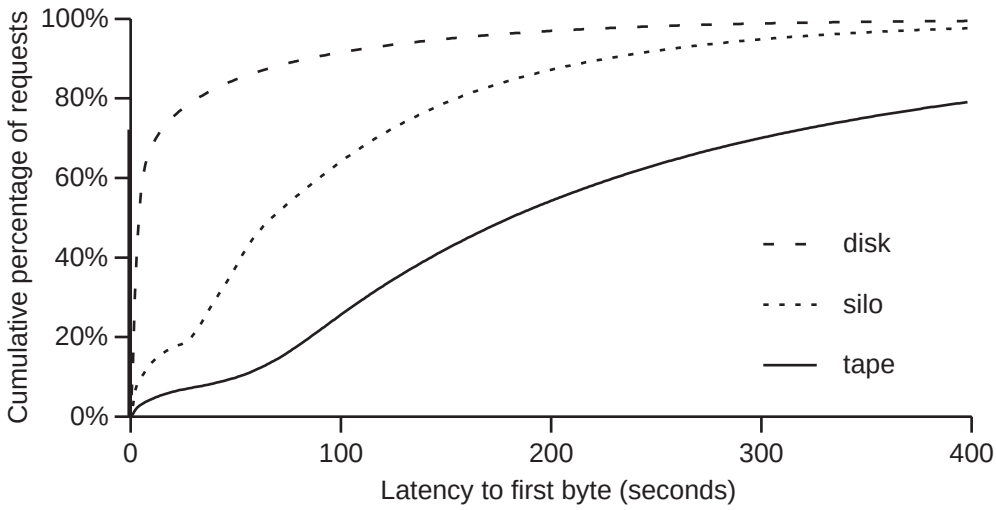

Figure 3-2. Latency to the first byte for various MSS devices.

This figure shows the total latency incurred in accessing several types of devices in the NCAR mass storage system. The disks in this diagram are those attached to the IBM 3090, not those in the Cray file system. The graph shows, as expected, that disks have the lowest latencies, followed by tapes in a silo and manually-mounted tapes.

Differences between manually loading a tape and having it fetched by the tape robot, however, are entirely caused by queueing delays. A tape silo can fetch a tape much faster than can a human operator. After subtracting off the queueing time exhibited by the disk (which is no longer than tape queueing time), the silo is approximately 2 to 2.5 times as fast as the manual tape drives at getting to the first byte. Since the tape silo tape drives are the same as the operator-loaded tape drives, this difference must come from the time to mount the tape rather than from seek time. The StorageTek 4400 ACS can pick and mount a tape in under 10 seconds; after subtracting off average queueing time for the disk, which is 25 seconds, the non-seek overhead for reading an automatically-loaded tape is 35 seconds. According to Table 3-2, tape accesses take 85 seconds on average, so the average seek is 50 seconds long. When the same analysis is applied to manually loaded tapes, the manual tape mounting time is found to be approximately 115 seconds, or about 2 minutes. This is quite good. However, as Figure 3-2 shows, 10% of all manual tape mounts were not completed within 400 seconds. Nearly all of the tape silo and disk requests were completed by this time. This is probably the biggest weakness of manual tape mounting — the very long tail of the mounting time distribution. While other data accesses will almost certainly complete in 5 minutes, manual tape mounts may take much

longer. Automatic computer access to mass storage data is thus valuable also for its consistency. It is difficult to gauge how estimate the response time of manually loaded tapes, making it harder to optimize system resources.

This is just a cursory analysis. There are several factors that were not considered which may affect these conclusions. In particular, queueing time for the tape silo may be different from queueing time for the disks. There are only a few tape robots in the silo, and each is tied up for several seconds with a tape load. If several tape loads come in close together, some of them will have relatively long queueing times. This does not happen with disk, as each disk is active for relatively little time with each request.

Another observation is the relation between latency to access the first byte and time required for the entire transfer. Both the tapes and the disks can transfer at a peak rate of 3 MB/sec, but the observed rates are usually closer to 2 MB/sec. As a result, the transfer times are similar for the two media. For tape, an average file of 80 MB will take 40 seconds to transfer. This is comparable to the additional 60 second overhead from using tape instead of disk. One possible way to improve perceived response time in the system would be to use cut-through, as in [39]. Under this scheme, a call to open a file returns immediately, while the operating system continues to load the file from the MSS and keep track of how far it has gotten. When future requests are made, the call returns immediately unless the requested data has not yet been read. This scheme works because applications typically do not read data as fast as the MSS can deliver it. Instead of delaying the application, then, it allows the application and file retrieval from the MSS to overlap. This system would be difficult to use in the current NCAR configuration, however, since the MSS is not seamlessly integrated with the local disk file system. The bitfile mover processes would have to have special communication protocols with the local file system to let it know how much of the file has been transferred. Nevertheless, it is a useful optimization and should be considered.

3.3.3. MSS Usage Patterns

Figure 3-3 shows the average amount of data transferred each hour of the day. As expected, activity is highest during working hours — from 9 AM to 5 PM. The variation in transfer rate, however, is almost entirely due to reads. The amount of data read jumps greatly at 8 AM when the scientists usually arrive, and slowly tails off after 4 PM as they leave. The fall is slower than the rise because most scientists are more likely to stay late than to arrive early. This suggests that most reads on the system are initiated by interactive requests, since reads peak when people are at work, while writes remain almost constant regardless of the number of humans requesting data. File request rate over the course of a day shows a pattern similar to that of data transfer rate.

The weekly data transfer rates, shown in Figure 3-4, have patterns similar to those in the daily averages. As expected, read activity is lower on the weekends, since there are fewer researchers around to initiate read requests. Write requests, on the other hand, experience little variation over the course of the week, as the Cray CPU runs batch jobs all weekend. There is a small increase in write requests during the day, indicating that users do actually make some write requests; however, the change is small relative to the flood of read requests that users generate.

Less data is transferred early Monday morning than on any other day. This low point can be attributed to two factors. First, the Cray was occasionally shut down early on Monday morning for maintenance, as that would cause the least disruption of normal work. Second, any idle time the Cray might have would be on Monday morning, as the queues from the weekend might have finished.

Over the two years the trace covers, the mass storage system received increasingly large amounts of work. The average data rate for each of the 104 weeks is shown in Figure 3-5. There are drops in read request rate around Thanksgiving and Christmas for both 1990 and 1991. Note, however, that write request rate does not drop on these holidays. In fact, write requests *increased* at the end of the year. This reinforces the con-

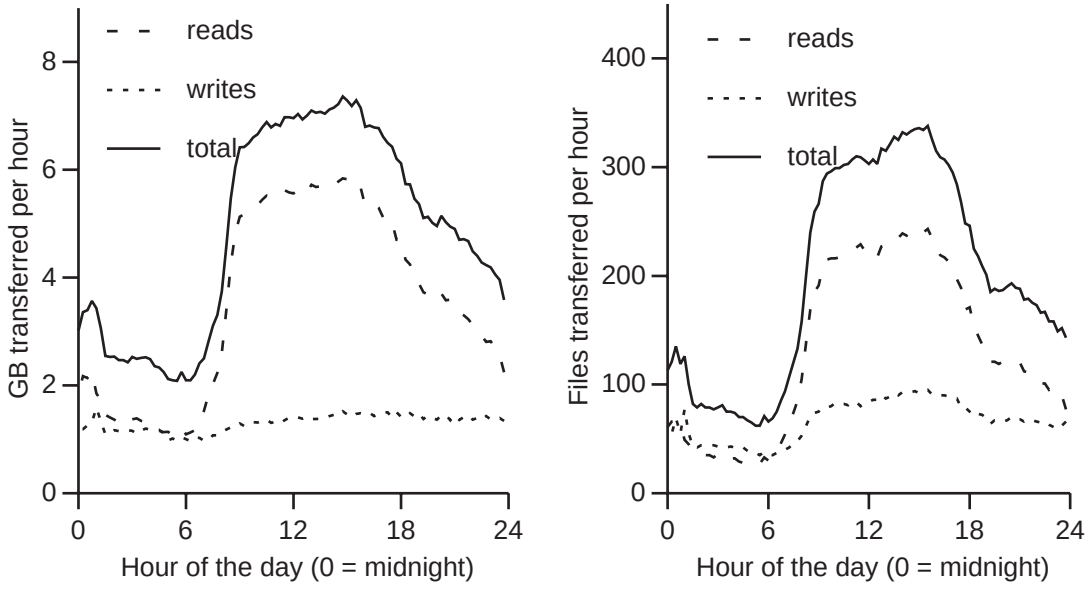

Figure 3-3. Daily variations in MSS access rates.

The graph on the left shows average transfer data transfer rates over the traced period for each hour of the day. The graph on the right is similar, but instead shows the average number of files transferred during each hour of the day.

clusion that reads are interactive while writes are requested primarily by batch jobs: the Cray does not take a Christmas vacation, while the scientists do.

The MSS data request rate increases over the period shown by the graph, but this gain is due almost entirely to increases in read requests. MSS write rate appears to be related to the speed with which the computer can generate results, while read rate is set by the number of users that want to read their data back. The lack of increase in write rate suggests that the Cray is already running at full capacity, and that researchers are simply using the machine more for tasks such as visualization of the results. A faster machine would then need a higher write rate to massive storage. There would be at least a corresponding increase in read rate, and it might be greater if the user community gets larger.

3.3.4. Interreference Intervals

Figure 3-6 shows the distribution of intervals between references to the MSS. Since about 3,500,000 files were referenced over a period of 731 days (approximately 6.3×10^7 seconds), the average interval between MSS requests was 18 seconds.

While the mean access time was 18 seconds, the median access time was only 1 second. In addition, 90% of all references followed another by less than 10 seconds. This distribution suggests that I/Os are clustered. There are several possible explanations for this. Since Cray files can be of (nearly) unlimited length, but files on the MSS cannot exceed 200 MB, clustering could occur since several files are accessed together by the same program. Another possibility is that there are really two distributions for intervals—those made by researchers' interactive requests, and those made by batch jobs. The interactive requests are very likely to be bunched together, since a researcher interested in day 1 of a climate model simulation will usually be interested in day 2, and the two days will probably be in separate files.

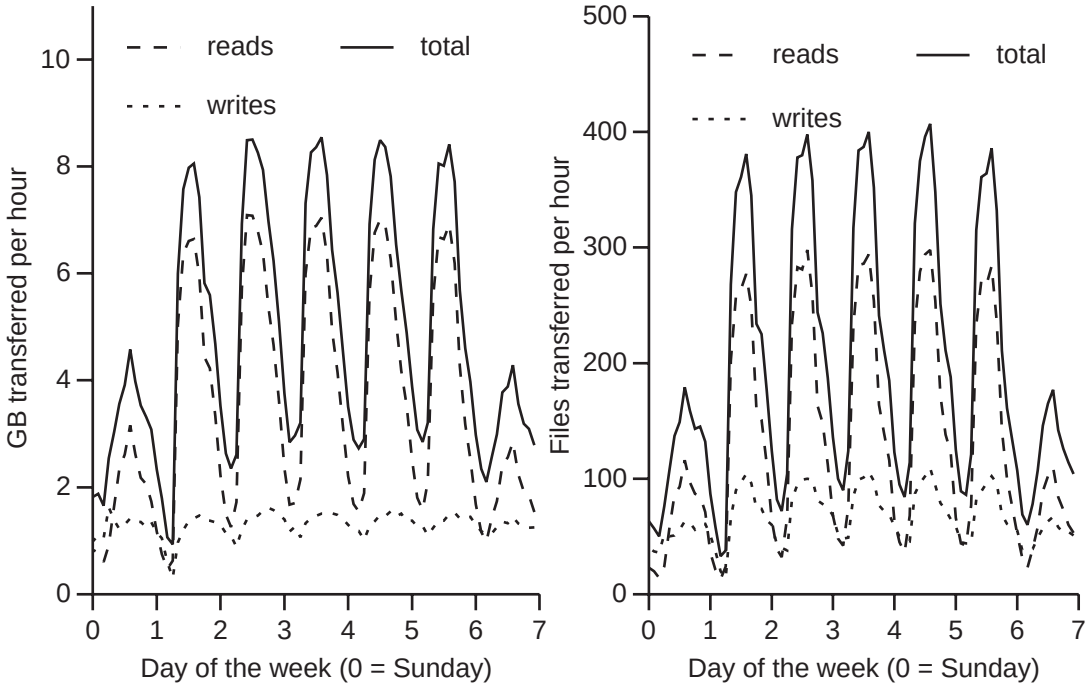

Figure 3-4. Weekly variations in MSS access rates.

As in Figure 3-3, the graph on the left shows the average data transfer rate, while the graph on the right shows the average number of files transferred per hour. Note, however, that the tops of the vertical axes are slightly higher than those in Figure 3-3, because the previous graphs averaged all days together. Weekdays, shown separately here, experienced higher transfer rates than weekends.

3.3.5. File Reference Patterns

Instead of counting all file references, this part of the analysis included at most one read and one write from any eight hour period. Since files on the MSS were explicitly referenced by Unix commands, some files were accessed many times in a short time. Since the NCAR system required a user to explicitly request a file stored on tape, making several such requests in a short period of time would result in a tape read for each request. In a system with a single name space and transparent access, this would not be likely to happen because the requested file would be cached on disk and returned to the user without a tape request. Thus, repeated accesses within a short period of time were filtered out because they artifically inflated tertiary storage access rates.

As expected, most files were not referenced often. Figure 3-7 shows that only 5% of all files are referenced more than ten times. 50% of the files in the trace were never read at all, and another 25% were read only once. Writes were slightly different — just over 20% of the files were not written during the trace period, but another 65% were written exactly once. Of course, these numbers add up to more than 100%: many files were read and written one time or less. In all, 57% of the files were accessed exactly once, and 19% were accessed exactly twice. Thus, only a quarter of the files were accessed more than two times. For this system, the median number of file references was one, contrary to [80], which reported the median to be two. Furthermore, fully 44% of all the files in the trace were written exactly once and never read. These files were “real” data, not checkpoints; files had to be explicitly written to the mass storage system, and it is unlikely that scientists would archive checkpoints. These numbers confirm the common belief that many files are written to a massive store once and never read again.

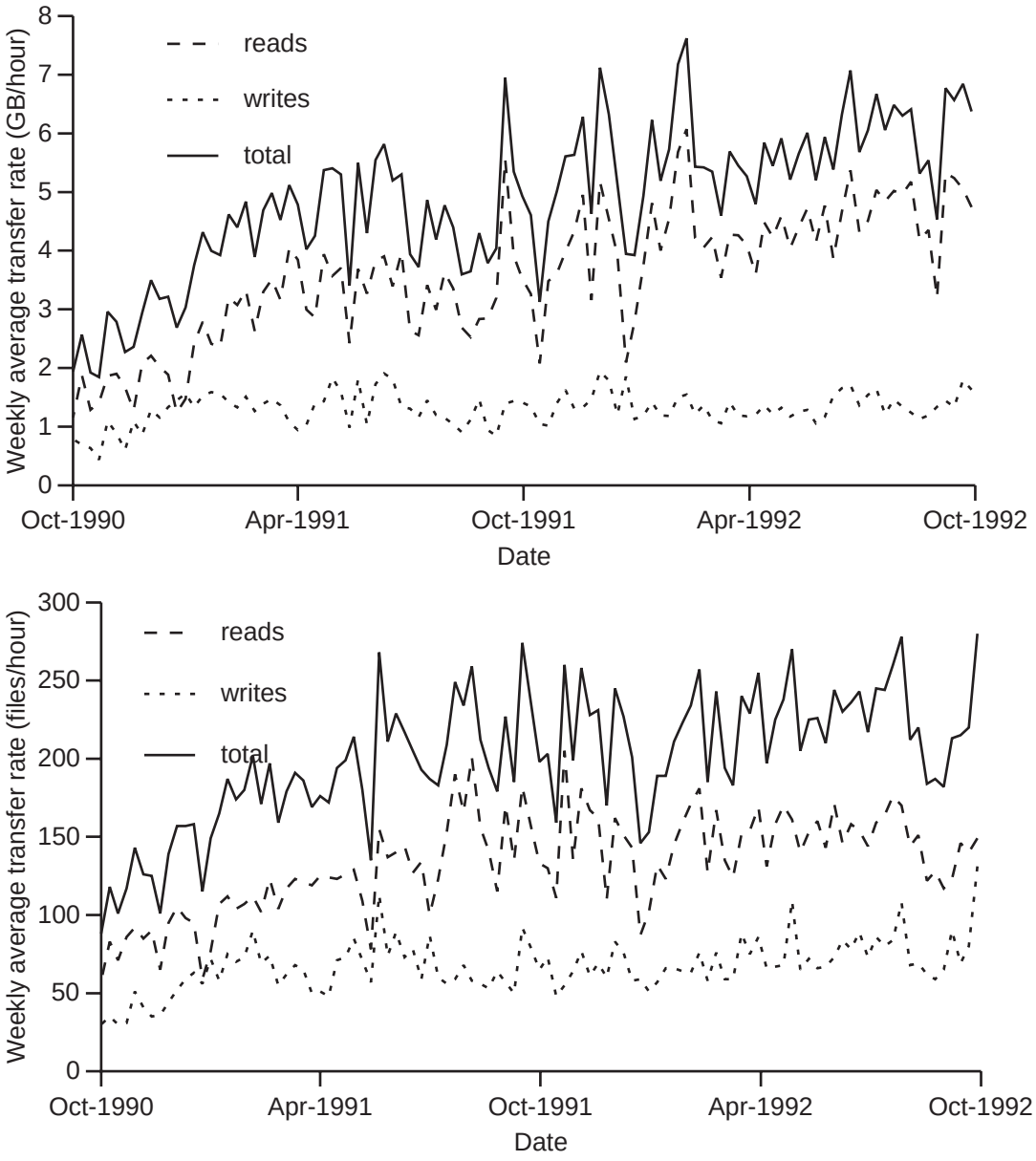

Figure 3-5. Long-term variations in MSS transfer rates.

The top graph shows the average weekly transfer rate in GB/hour, while the bottom graph shows files/hour. Only the weekly average is graphed, as the variation within a week is so high that the graph would be unreadable. Holidays such as Christmas, Thanksgiving, and Memorial Day are visible as temporary dips in the access rates.

Figure 3-8 shows the distribution of time intervals between references to a given file, called *interreference intervals*. Long interreference intervals mean that a file is referenced infrequently, while short intervals indicate many accesses over a short period of time. As Figure 3-8 shows, interreference intervals were short. This means that, for files that were rereferenced, the second access came soon after the first.

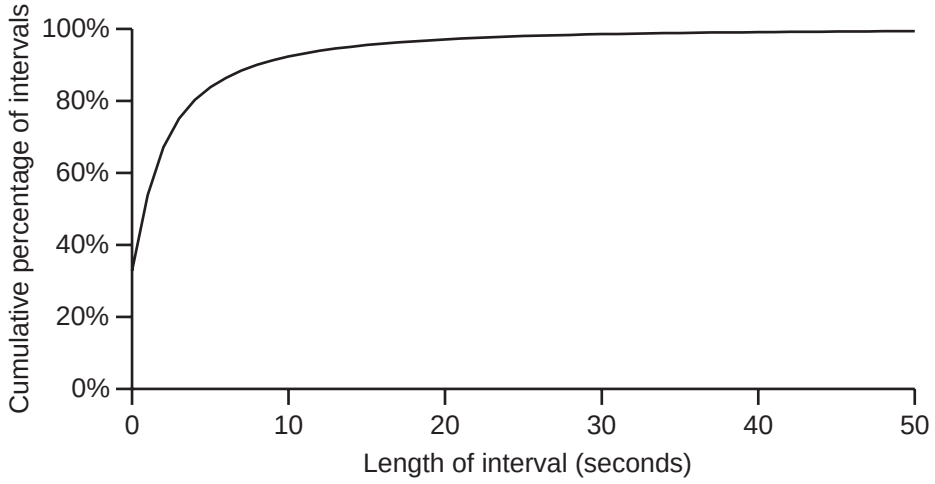

Figure 3-6. MSS interreference intervals.

Lengths of intervals between Cray Y-MP references to the mass storage system. Interval lengths of 400 seconds or shorter (99.98% of all intervals) were recorded, but the graph above only shows those up to 50 seconds long (99.36% of all intervals).

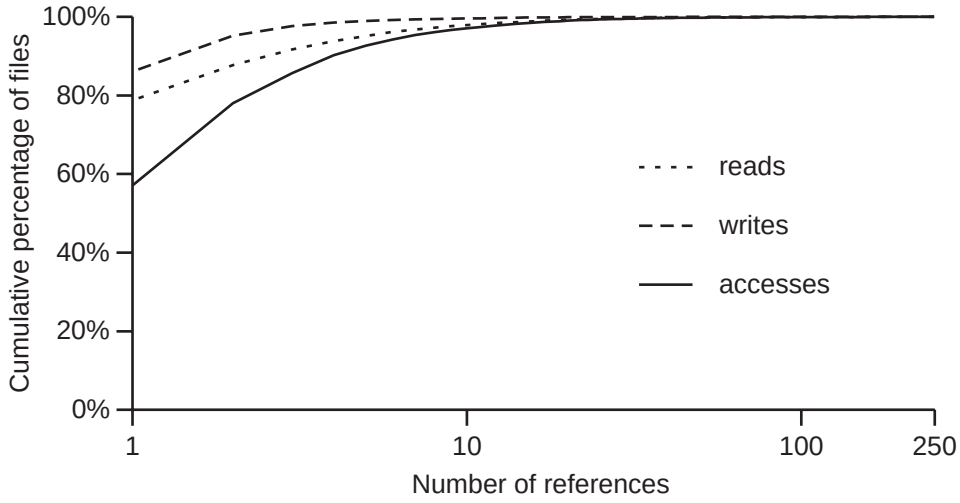

Figure 3-7. File reference count distribution.

This figure shows the total reference count of each file referenced during the trace period, broken into reads, writes, and total references for each file. Each line in the graph is a cumulative histogram of the files referenced *n* or fewer times (at *n* on the horizontal axis). During the trace period, 50% of the files were never read, and 21% were never written. All files were accessed at least once, however.

However there were still some files that were referenced more than a year after the previous reference to them. These references could not be easily predicted, so it is not sufficient merely to use prediction to improve access times. *Instead, the latency for random requests must also decrease*.

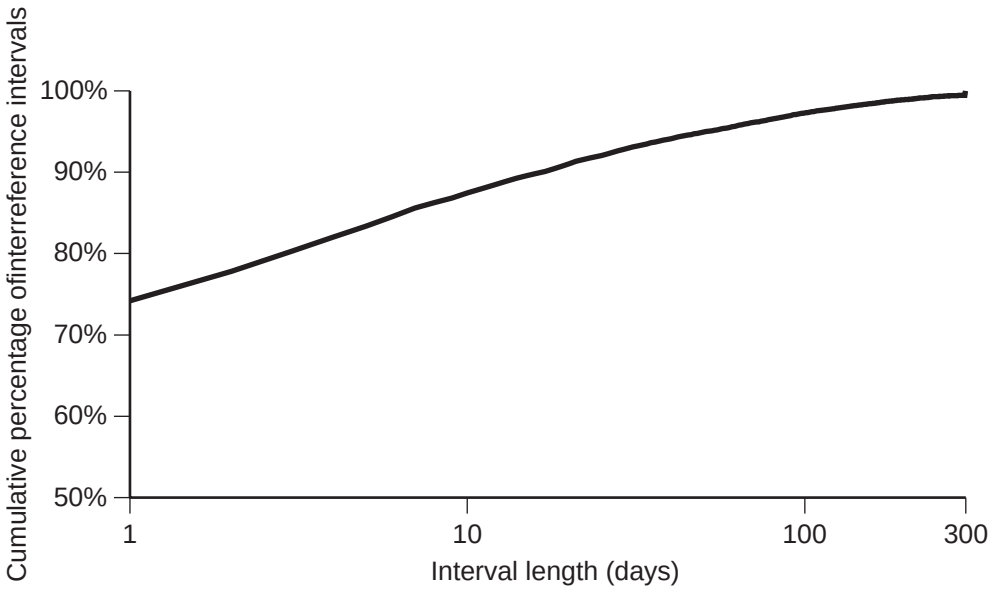

Figure 3-8. MSS file interreference intervals.

This figure shows the distribution of lengths of interreference intervals. Many files were referenced only once; these files had no interreference intervals. In addition, only one reference per eight hour period — the first — was included in the data in this graph. This graph only shows distribution for intervals up to 300 days long. 1% of the intervals were longer than 300 days.

3.3.6. File and Directory Sizes

The dynamic distribution of file sizes transferred between the MSS and the Cray is shown in Figure 3-9. In this graph, a file is counted once for each access to it. The distributions of files read and files written are similar, though there is a small jump in file writes at approximately 8 MB. However, 40% of all requests are for files 1 MB or smaller. Since reads are more likely than writes to be initiated by a human user (as Section 3.3.3 shows), this graph suggests that performance on small file reads in a migration system would be especially important. Such small files make up under 1% of the total data storage requirement, so it seems wise to store these files on inexpensive commodity disks rather than on tape. If magnetic disk would be too expensive, an optical disk jukebox could provide low latency to the first byte and high capacity.

The distribution of file sizes on the MSS during the trace period is graphed in Figure 3-10. In it, each referenced file is counted exactly once, regardless of the number of times it was accessed. The graph shows that, while about half of the files are under 3 MB, these files contain 2% of the data. Algorithms that take file size as an argument could use this fact to simplify their bookkeeping, as all files below a threshold size could be considered equivalent when computing space-time products. Since most files are below this size, the algorithm should run much faster.

Directories also tended to be small. Figure 3-11 shows that 90% of the directories had 10 or fewer files, and 75% had only zero or one file. Even so, over half of all files and data were in large directories that contained more than 100 files. The size and number of directories is very important, as many current systems do not archive directories or file metadata. With over 130,000 directories and 900,000 files, the NCAR system needs to store gigabytes of metadata on disk. Future systems must be able to move this information to tape, especially since over 40% of the metadata describes files that will never be accessed again. The RAMA file

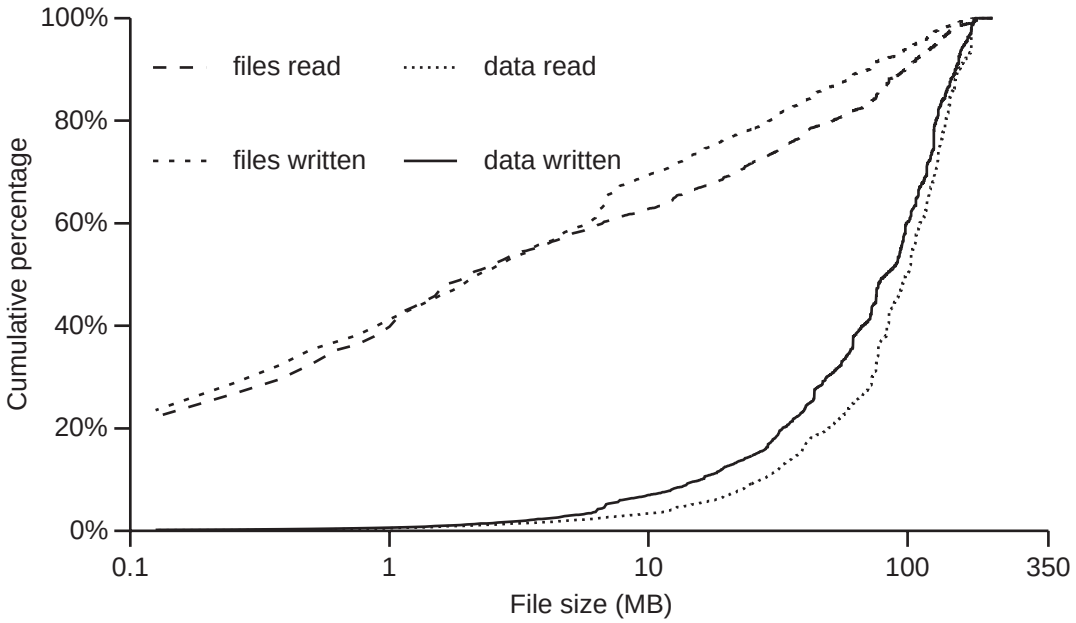

Figure 3-9. Size distribution of files transferred between the MSS and the Cray.

In this figure, a file is counted once for each time it is requested. This shows the sizes of files actually transferred between the Cray and the MSS, as opposed to the size distribution of files stored on the MSS. Note that the largest file the MSS can store on tape is 200 MB, limited by the capacity of the tape cartridges.

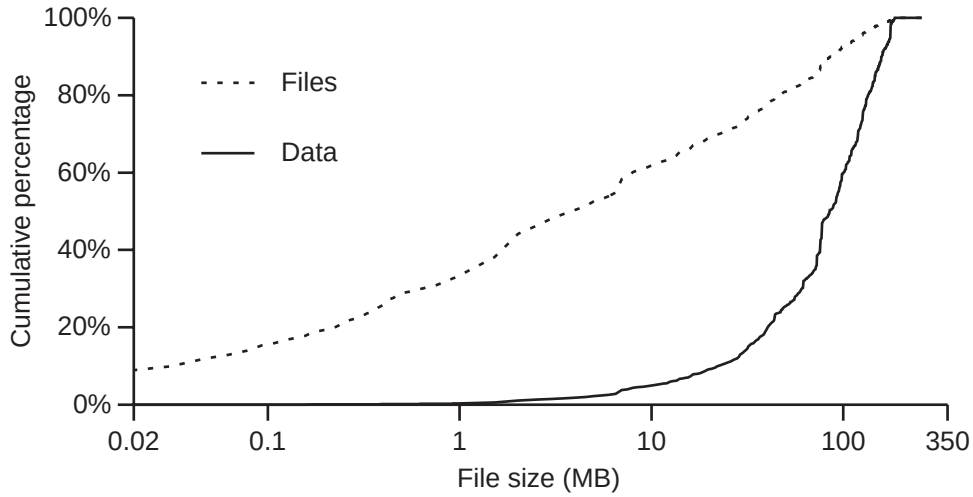

Figure 3-10. MSS static file size distribution.

While 50% of the files on the MSS are relatively small (under 1 MB), very little data is in these files. Most of the data on the MSS is stored in files longer than 30 MB, and thus is stored on tapes rather than on disk.

system described later in this thesis allows directories to move to tape, thus eliminating the problem of storing the metadata for terabytes of files on secondary storage.

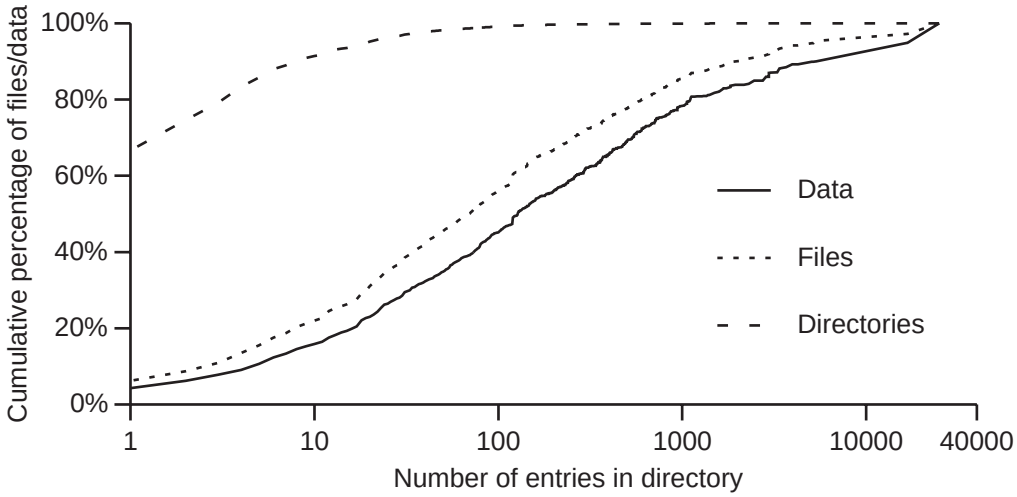

Figure 3-11. Distribution of data and files by directory size.

Most of the data is in files stored in large directories — those with over 100 entries contain over 60% of all the data and files on the MSS. This is surprising, since only 2% of the directories have 100 entries or more.

3.4. File Migration Algorithms

The NCAR trace data has several implications for future file migration algorithms. The system studied here is quite different from that examined in earlier studies [31,81]. While file access patterns have not changed radically from 1980 to now, and most files are still written and subsequently read one or fewer times, the files themselves are, on average, about 100 times larger. The average file today is sufficiently large that transfer time is an important part of the time needed to fetch a file; this was not true in 1980. Additionally, the 1980 system stored 25,000 files, as compared to the nearly one million files on the NCAR system by October, 1992.

The NCAR system uses two different migration algorithms — one for moving files between the Cray and the MSS, and the other for relocating files on different media within the MSS. Moving files between the Cray and the MSS is entirely manual, so there is no choice in the “algorithm” involved. However, using automatic migration between the Cray and the MSS could still save many file requests. About one third of all requests came within eight hours of another request for the same file. Often, these accesses are generated by batch job scripts which must read or write files on the MSS. If several of these scripts are run at about the same time, the Cray must make a separate request to the MSS for each script; it has no way of keeping track of multiple references to the same file. Better integration of the MSS with the Cray would fix this problem by allowing the Cray file system to recognize successive references to a file recently fetched from the MSS.

Another change since the studies done around 1980 involves large files. Previous algorithms optimize for low seek time and ignore transfer time. For multi-megabyte files, however, transfer time dominates the time needed to access a file. On magnetic disk, seek time is far lower than transfer time for megabyte-sized files. Even for robotic tape, however, seek time is comparable to transfer time. A StorageTek robot can load a 3480 tape in under 10 seconds; the drive can transfer 20 MB in this time. The standard algorithms all make the assumption that the retrieval cost is the same for all files (though the storage cost may not be). However, this is not true for the NCAR system. A 1 MB file can be read from tape in 0.3 seconds, while a 200 MB file requires over 65 seconds. New algorithms will have to consider this disparity in access time. The NCAR system already does this by storing smaller files on magnetic disk and larger files only on tape. In this way,

small files do not suffer the latency penalties of tape. Large files, on the other hand, must wait for a tape to be loaded. However, their transfer time is long enough that the added delay of loading a tape is not as noticeable. The dividing point between storing files on disk and storing them on tape is a subject for future research; however, it is likely that the switchover point will be a function of tape seek speed and transfer rate.

Previous algorithms also make little distinction between reads and writes, primarily because their trace-gathering methods did not allow them to distinguish a read access from a write access. However, this difference is crucial for a file migration algorithm. The read/write ratio to the MSS at NCAR is 2:1, contrasting with conventional wisdom that an MSS services more writes than reads. Additionally, humans must wait for the results from reads, while users would not need to wait for writes to tape to complete. This suggests that an algorithm should not wait until it is absolutely necessary to free up space; instead, it should write data to tape relatively quickly, and then mark the file as "deleteable." Since files would be written lazily, their placement on tertiary media could be optimized, making future reads run faster. A mass storage system should be optimized to make read access to files faster at the cost of requiring more work for writes. This will make the system seem faster to its users at little additional cost.

3.5. Conclusions

This analysis of file movement between secondary and tertiary storage at a supercomputer Unix site provides several important hints for designers of file migration systems. First, humans wait for reads, while computers wait for writes. Any migration policy should consider this, and optimize for reading. The write rate is relatively steady over time, while reads vary greatly. Thus, migration algorithms should move files to tertiary storage whenever resources (tape drives, etc.) are available, and use the extra space to prefetch files which might be read shortly. This study also shows the need for designers of high-performance file systems to integrate the MSS more tightly with high-speed secondary storage. 30% of the file migration references could have been avoided by using implicit rather than explicit requests to the MSS. This would be possible only if the Cray's file system were tightly integrated with the MSS.

Files have become larger and more numerous since the early 1980s. In late 1991, there were over 900,000 files on the MSS at NCAR averaging over 25 MB each. On the other hand, their reference patterns have not changed much. File rereference rate still drops off sharply after the first few days, though it does level off soon thereafter. Files are also infrequently rereferenced; more than half of the files were only accessed once in two years. Again, this suggests that files can be migrated to a less costly storage medium if they are unreferenced for only a few days.

The NCAR system appears to be a typical large Unix-based scientific computing center. Thus, the analysis in this chapter will help system architects design hardware and software best suited for storing and rapidly accessing the terabytes of data that such systems must store. While reference patterns for these data have not changed much in the last decade, more files, larger files and new tertiary storage technologies will require new mass storage systems and new migration algorithms to run them.

The analysis also showed the necessity for a tight coupling between the tertiary storage system and the secondary storage system. This is necessary to allow the file system to optimize file placement and transfer, especially for environments where data is shared between researchers.

A file system needs several features to facilitate close cooperation between secondary storage and the mass storage system. First, it must maintain a single namespace for the entire storage system rather than allowing files to have different names on disk and tertiary storage. In this way, a user making a file request cannot know where the file resides, allowing the system to transparently cache and migrate files between high-speed disks and the MSS.

The file system must also permit metadata to migrate to tertiary storage. Even if metadata requires only 0.2% of actual storage space, 50 TB will still use 10 GB of secondary storage. This is space that should be better used for faster access to files, not storage for metadata that is likely to be used once per year. Migrating metadata to the MSS is also necessary in a computing center with more than one secondary storage system, since a file whose data has moved to tertiary storage could be read by any computer in the system. Thus, metadata pointing to files on tape should not reside on any secondary storage; rather, it should be stored near the taped files themselves.

Since a high-performance file system may prefetch files from tape without an explicit application request, it will be unable to use program hints to use the optimal layout on the many disks in the system. Many current file systems provide an order of magnitude improvement in performance if the application supplies data placement hints to the file system. If a file is prefetched, though, such hints will not be available, as the program that will eventually read the data may not be the same one that wrote it. A file system on disk that is well-integrated with tertiary storage must therefore provide high-bandwidth file access without the help of hints from an application.

A parallel file system for scientific computing must thus satisfy the following requirements, based on the study presented in this chapter and the scientific computing demands described in Chapter 2:

- High bandwidth performance on large files, regardless of the mapping of file blocks to disks.
- Low latency on small files to allow integration with workstation file systems.
- Unified name space across secondary and tertiary storage.
- Transparent migration of both file data and metadata between secondary and tertiary storage.

The remainder of this thesis focuses on the design and simulation of a file system for parallel machines that integrates scalable secondary storage and tertiary storage, using the design suggestions listed above. This file system would make an ideal platform for real-world testing of file migration algorithms for scientific computing.

4 RAMA: a Parallel File System

This chapter describes the design of the RAMA (Rapid Access to Massive Archive) parallel file system and the reasoning behind the design. The RAMA file system is designed to provide high-bandwidth access for large files while not sacrificing performance for small files. It is thus best used for scientific computing environments, which require many files tens of megabytes to gigabytes in length.

RAMA takes advantage of recent advances in disk and network technology by placing a small number of disks at each processing node and pseudo-randomly distributing data among those disks. Pseudo-random distribution, achieved using a hash function to compute the location to place file blocks, provides two main advantages: good performance across a wide variety of workloads without data placement hints and scalability from fewer than ten to several hundred node-disk pairs. RAMA requires a fast interprocessor network to overcome the slight latency disadvantage of not placing data "near" the node that will use it, as discussed in Section 4.1.2. While physically small disks are not necessary for RAMA, they reduce hardware cost and complexity by allowing disks to be mounted directly on processor boards rather than connected using relatively long cables. Currently, 2.5" disks can provide up to 1 GB of storage; thus, a 128 processor MPP could have a 128 GB file system using just a single disk attached to each processor.

As Chapter 3 shows, file migration is an important feature of file system that support scientific computation. Many file systems for scientific computation do not support tight integration with tertiary storage, causing unnecessary requests to tertiary storage and inefficient use of secondary storage space. RAMA, however, is designed as a cache on disk for a multi-terabyte tertiary storage archive. Many design decisions in RAMA, such as pseudo-random distribution, would be difficult to implement without tertiary storage. More than that, though, RAMA supports a uniform view of the file system as proposed by the Mass Storage System Reference Model [13]. An application using the RAMA file system need never know whether data is on disk or a slower device when it is requested, since the file on disk has the same identifier as its image on tape. RAMA can transparently fetch any file blocks the application needs, perhaps moving only the data that must be moved instead of the entire file.

This chapter examines the decisions we made while designing RAMA, and mentions some of the design alternatives. As with any file system, the RAMA design must provide schemes for allocating and laying out data and metadata. Additionally, it must provide mechanisms to translate logical file block addresses to physical disk block addresses and reclaim disk space that was used for a deleted file. All of these operations must take file system consistency into account. RAMA, like other file systems, must insure that the it never incorrectly assigns data to a file. Additionally, RAMA must maintain state for each disk block, insuring that blocks always either free or allocated but not both or neither. Since RAMA will be used for scientific computation, it must also provide high-bandwidth access to data in the file system. Algorithms on MPPs often have many nodes accessing a single file, requiring RAMA to read and write different sections of a file without a "master" node for the file.

RAMA's use of a basic structure called the *disk line* allows simple schemes to address all of these issues in file system design. RAMA pseudo-randomly assigns blocks to disk lines using a hashing algorithm known at each node. As a result, any node in the MPP can narrow a block's location to a single disk without the aid of a central "directory" node. This scheme allows all of the nodes in an MPP to access a single file without any bottlenecks at any single node. Since blocks from a single file are distributed to many disks, large files can be read at high bandwidth. Spreading data to disks by block rather than smaller units also allows RAMA to deliver the high concurrency access for small files necessary in a workstation environment. RAMA also provides good crash recovery, as each block's state is stored only in the table of contents of its disk line. Since a block's descriptor includes its file identifier and offset as well as its state (free, clean, or dirty), this system eliminates the need for file system consistency checkers such as `fsck` [55]. This chapter explains RAMA's use of disk lines and presents the schemes RAMA uses to access and manipulate data on disk.

The three chapters after this one will discuss a simulation study of the RAMA file system described in this chapter. Chapter 5 covers the design of the RAMA simulator and describes the workloads used in the simulation experiments. RAMA's sensitivity to changes in technology, different configuration parameters, and average file size are studied by simulation in Chapter 6. Next, Chapter 7 compares RAMA to a simulated striped file system. This comparison uses several workloads to compare the two file systems' effects on application execution time, network performance, and disk load. It will show that RAMA provides comparable bandwidth to an optimally-striped file system while far outperforming a suboptimally configured striped file system.

4.1. File System Design

This section describes the design of the RAMA file system. Section 4.1.1 describes the metadata RAMA must maintain for each file and each block in the file system. The layout of data and metadata on disk is then covered in Section 4.1.2.

The RAMA file system is designed to run on MPPs with up to hundreds of processor nodes, each with a closely-associated disk, as shown in Figure 4-1. This hardware differs from that of many modern MPP systems, shown in Figure 4-2, that attach many disks to just a few nodes in the MPP. Unlike conventional file systems that stripe data in a regular pattern across some or all of the attached disks, RAMA uses a hash algorithm to distribute data pseudo-randomly to all of the disks. This distribution of data gives RAMA two major advantages: regular access patterns do not slow the file system down, and multiple nodes accessing different parts of the same file do not require a central synchronization node. These effects are discussed in Chapter 7.

4.1.1. File System Information

RAMA files, like those in most file system, are divided into blocks. A file block in RAMA may be any size. Previous experience [2,67] indicates that sizes between 1 KB and 16 KB are best for workstation workloads using current commodity disk technology, while blocks up to 32 or 64 KB are used in supercomputer file systems with large files and fast, expensive disks. Additionally, both workstation and supercomputer file systems attempt to allocate sequential blocks from a file consecutively on disk [56].

Each file in RAMA is identified by a 64-bit unique identifier called a *bitfile identifier (bitfile ID)*, as defined in [13]. Any active block in the entire file system can therefore be uniquely identified by the bitfile ID of the file it belongs to and its offset within that file. Further, bitfile IDs may be chosen randomly — they need not be sequential or otherwise related. RAMA can thus avoid reallocating bitfile IDs for files that have been moved to tertiary storage, simplifying the directory structure.

File blocks in RAMA may be in one of three states—clean, dirty, or free. Free blocks are just that. Data can always be written to an available free block. Clean blocks are those that have not been modified since they

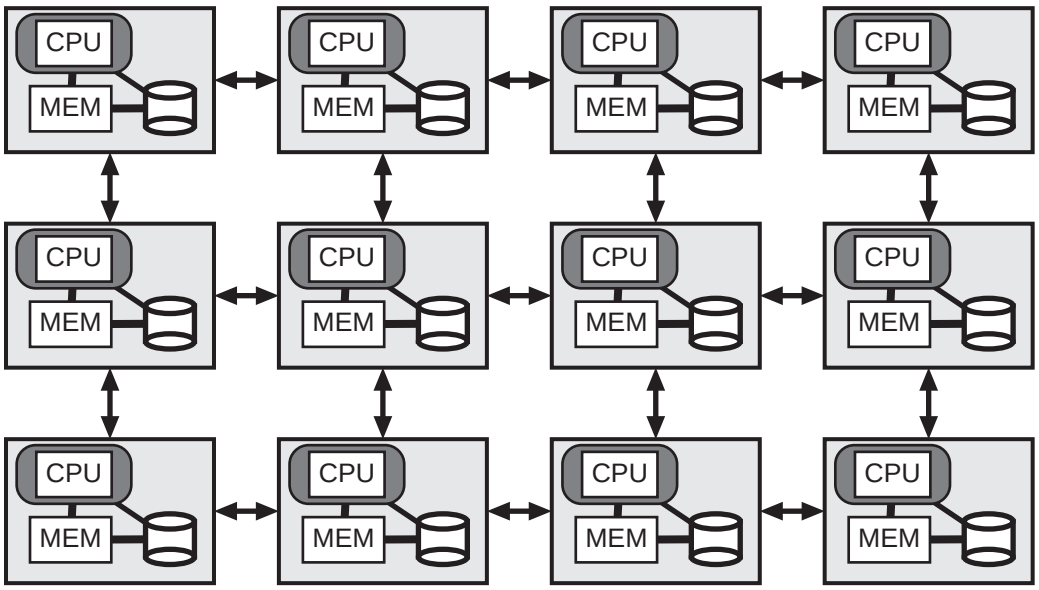

Figure 4-1. Typical hardware running the RAMA file system.

The RAMA file system is designed to run on MPPs with one or more disks attached directly to each node. Traditional MPPs, shown in Figure 4-2, replicate the CPU-memory module. RAMA, on the other hand, proposes that the replicated module be CPU-memory-disk. In this way, total disk bandwidth scales with the number of MPP nodes without requiring a faster connection to more disks.

were written to tertiary storage. This includes both blocks that have been written back to archive and those that have been retrieved from tertiary storage but not modified. If there are no free blocks, clean blocks may be reclaimed for new data. A copy of data in these blocks exists elsewhere, so they can be retrieved if needed later. Finally, dirty blocks are immune from overwriting. As a result, dirty blocks must be converted to clean or free blocks faster than the overall write rate. This simply means that migration from disk to tertiary storage must, on average, be faster than the rate that long-term data is created. Migration managers, described in Section 4.2.3, move data from disk to tertiary storage, thus converting dirty blocks to clean blocks.

Metadata in RAMA is divided into two types — *positional* metadata and *intrinsic* metadata. Positional metadata is information that RAMA needs to transform a bitfile ID and block offset to a physical block number on a specific disk. It is particular to RAMA, as the same file stored on a tape could certainly use different structures. Intrinsic metadata includes information that must be kept for a file regardless of the medium on which the file is stored. Intrinsic metadata includes attributes such as file length, modification and access timestamps, and file permissions that belong to a file and must move to tertiary storage with a file.

In an environment with large files such as those for which RAMA is designed, the two types of metadata are used differently. A typical MPP program will open a large file once and perform thousands or millions of read and write operations. Intrinsic metadata is used primarily when a file is opened or closed, and thus is not a main determinant of performance. Positional metadata, on the other hand, is used in every read and write, and greatly affects performance. The RAMA metadata scheme must support accesses from many nodes to different parts of a single file without requiring a central node to coordinate any modifications to the positional metadata. If this criterion is not met, performance cannot scale because all file system requests

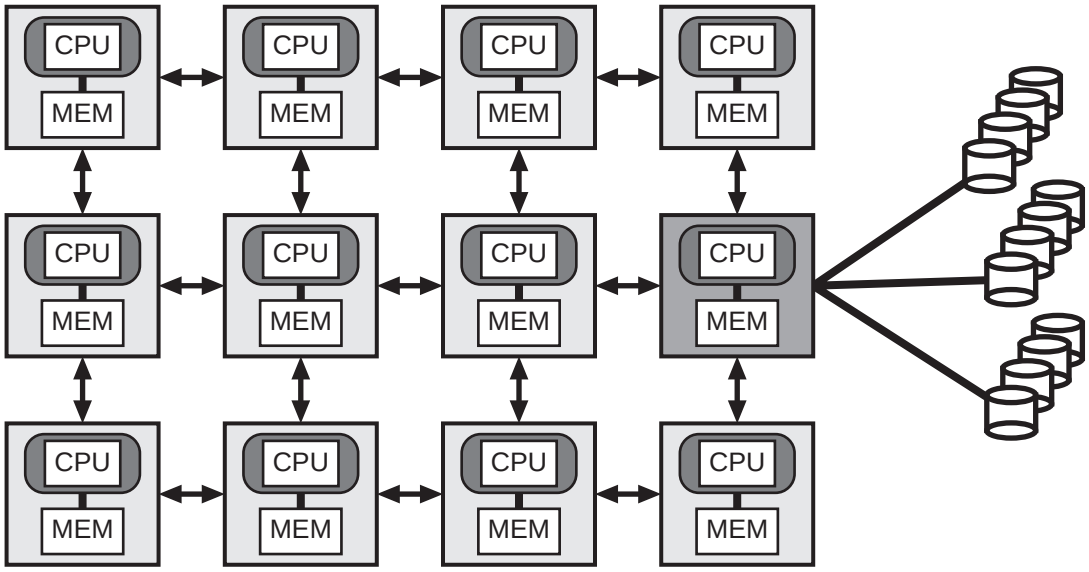

Figure 4-2. Typical hardware running conventional MPP file systems.

This figure shows the hardware that many MPPs, such as the Intel Paragon and Cray T3D, use to integrate disk storage with processors and memory. A small number of I/O nodes have high-bandwidth connections to all of the disks in the MPP. In the diagram above, a single node (the more heavily shaded one) connects to all of the disks the MPP uses. All disk requests must then go through this node.

must go through a single node. As later sections in this chapter show, the RAMA design is scalable because individual nodes use a hash function, not a central directory, to compute the location of the data they read and write.

4.1.2. Data Placement in RAMA

The data layout in RAMA is motivated by several principles. First, every node in the MPP must be able to transfer a file block specified by a (*bitfile identifier, offset*) pair by only talking to the server node that stores the data. Once a file is opened and its bitfile ID is known, a client node should not need to access a central "control" node for the file to read and write file data. Vesta [18,19] supports this by distributing information to each node detailing the way in which a file is striped. RAMA goes a step further by using a hashing algorithm to determine where a specific file block is stored, a method also used in [3] and [17]. The algorithm takes a bitfile ID and block offset as inputs, and is the same for all nodes in the file system. Since the algorithm is relatively easy to compute and requires no per-file information that might be stored on an individual node, it fulfills the criterion of independent access.

4.1.2.1. File Block Placement

The hash algorithm in RAMA is actually used to determine the *disk line* within which an individual file block resides, rather than just the node that stores the block. A disk line is similar to a line in a memory cache. It consists of the file blocks themselves and a *line descriptor* similar to the tags in a memory cache, as shown in Figure 4-3. The line descriptor acts as a "table of contents" for the disk line, keeping a list of *block descriptors* for the blocks in the line. Each block descriptor holds the bitfile identifier (64 bits), file

block number (32 bits), a timestamp used for migration (28 bits), and several flags (4 bits) encoding the block's state for as single file block in the disk line. These sizes were chosen because they provide a sufficiently large name space while keeping block descriptor size small. Longer bitfile IDs and block offsets are easily implemented at a slight cost in disk storage space. The line descriptor also holds a free map for the entire disk line and a separate free map for intrinsic metadata, described in Section 4.1.2.2. The number of file blocks a disk line contains is a system-wide parameter — fixed for all disks in the system — whose typical values range from a few hundred to a few thousand file blocks. Since a disk line is 400 KB to 20 MB long, a single disk in RAMA holds many of them.

This scheme has a slightly higher overhead than that of the Unix Fast File System [54], as each block requires 16 bytes of line descriptor space, compared to the 4 bytes required in FFS. However, for 8 KB blocks, positional metadata consumes less than 0.2% of total disk space in RAMA. This percentage will drop further as future RAMA file systems use larger file blocks to better utilize faster disks.

The disk line design is an intermediate choice between two extremes. One possibility would be to have the hash algorithm only determine the node, and allow each node to manage the placement of data on its own disk. While this scheme allows more flexibility for each node to manage the data it is providing, it leaves the difficulty of finding randomly-selected bitfile IDs on a disk. At the other extreme, the hash function could produce a physical block identifier that uniquely identifies a specific block on a specific disk. This approach suffers from two problems. First, there is no guarantee that the block is valid. Since a file may not actually be on disk, some mechanism must exist to make sure that a particular block mapped by the hash function does indeed contain valid data. Second, and more serious, the hash algorithm could map two different file system blocks to the same physical block. There would then have to be some form of conflict resolution. Rehashing would not be acceptable if it occurred often, since it would slow down the file request by making additional disk I/Os to find the desired block.

Allowing each block to be accessed independently provides excellent parallelism; however, large sequential reads and writes from a file must be broken into smaller reads on many disks. Most file systems optimize data placement so large file accesses can be completed with few seeks. This is implemented in RAMA with a minor modification to the hashing algorithm. Instead of assigning each file block to a different disk line, the algorithm can be changed to map several consecutive blocks of the same file to the same line by dividing the block's offset within the file by the *sequentiality* parameter before sending it to the hash function. If *sequentiality* is 16, for example, all blocks with offsets between 16 and 31 will send 1 to the hash function, generating the same hash value for all of them. *Sequentiality* can be adjusted to trade off between faster sequential access and conflict problems. If too many consecutive blocks from a single file go to the same disk line, the line will be unable to hold much other data that hashes to the same line. This would degrade performance by forcing RAMA to store the excess blocks for that line on tertiary storage. Additionally, having too many consecutive file blocks in the same line reduces performance on medium-sized files, as a file request of a given size results in requests to fewer disks with more data per disk. On the other hand, too few sequential blocks in the same line reduces performance by requiring smaller nonsequential transfers and more seeks. The optimal choice for *sequentiality* depends both on the workload and on the disk characteristics. Faster disks and larger file requests achieve higher bandwidth with more consecutive file blocks in a disk line; however, workstation workloads in which small files are common perform best with fewer consecutive blocks per disk line. Section 6.2.1 discusses the effects of changing the sequentiality parameter to the hash function. The effect of varying the total number of blocks in a disk line, however, is beyond the scope of this dissertation and is a subject for further investigation.

Another factor in data placement is the arrangement of data blocks within a disk line. Naive placement would scatter sequential blocks from the same file around the disk line, which holds hundreds of data blocks. This provides suboptimal performance, since reading sequential blocks from a file would require rotational and head switch delays. This difficulty can be easily overcome by arranging data in a disk line so

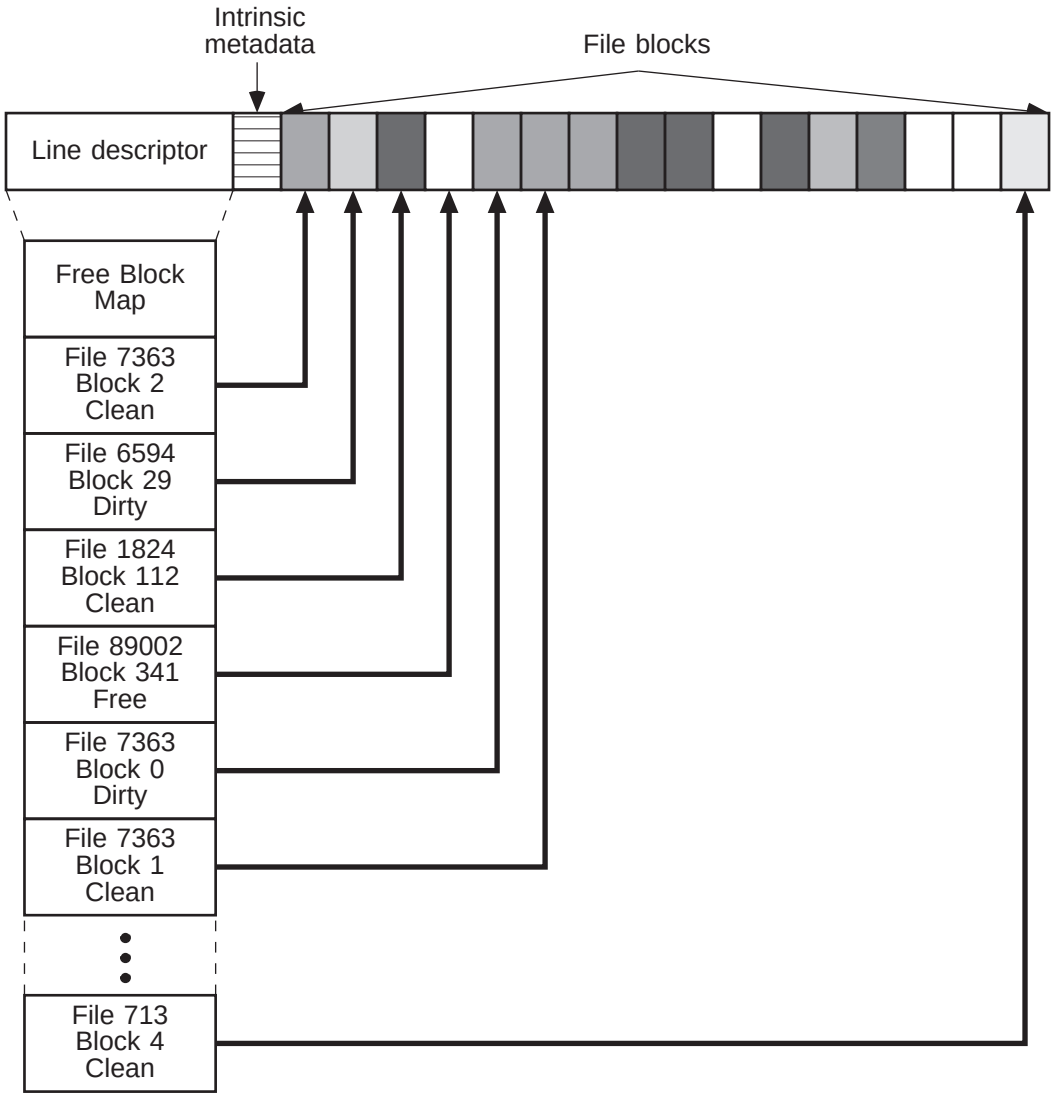

Figure 4-3. A RAMA disk line and line descriptor.

The disk line is a basic element of the RAMA file system. It contains a table of contents — the line descriptor — and hundreds to thousands of file blocks. Each file block in the line corresponds to exactly one entry in the line descriptor. That entry identifies the block and provides additional information such as what state the block is in. Section 4.2 describes RAMA's operation and discusses how the file system uses disk lines to store data.

that consecutive blocks from a file are stored contiguously on disk. This does not introduce additional problems because placement within a disk line is governed only by the server for that disk line, and need not follow any other constraints. Section 4.2 further describes this process of reorganization.

4.1.2.2. Intrinsic Metadata Placement

RAMA supports two different design choices for intrinsic metadata placement. The first choice groups intrinsic metadata for many files together into blocks near the start of a disk line, storing the metadata in

much the same way as the BSD FFS uses cylinder groups [54]. Another choice, however, would store the metadata in the same block as the first few bytes of the file itself. These choices are shown in Figure 4-4.

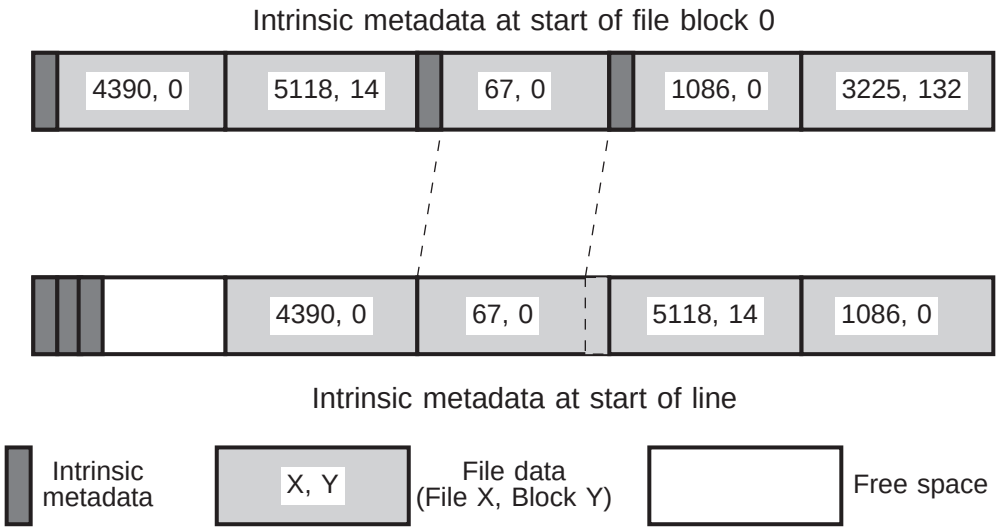

Figure 4-4. Intrinsic metadata placement options.

RAMA supports two different placement options for intrinsic metadata. First, intrinsic metadata could go into the first block of any file on disk. This option, shown at the top of the figure, would skew the start of every file block in the file by the size of the metadata. Since many programs assume that file blocks start at multiples of the file block size, this option could degrade performance by poorly dividing the file. The second option, at the bottom of the figure, would place all of the intrinsic metadata for each disk line in a few blocks at the start of the line. Any files whose first block is in the line would store its metadata at the start of that line. The actual location of the metadata within the dedicated metadata blocks could be found either by searching through each block or, better, by encoding its position in the line descriptor entry for the first block of the file.

While each has its own advantages and disadvantages, they both share several characteristics. Both keep intrinsic metadata near the start of file's actual data on disk, reducing seeks. Storing intrinsic metadata in the first file block is attractive for moving files between secondary and tertiary storage. When the first block of a file is reused, either because the file is deleted or because it has been migrated to archive, the space for the intrinsic metadata for the file is also reclaimed. However, putting intrinsic metadata in the first file block preceding the file data creates other problems. In particular, file blocks no longer start at even multiples of the block size. Instead, the first block contains less than a "full" block of data, and future blocks have their starts offset by the size of the intrinsic metadata.

Grouping intrinsic metadata from many files into a few blocks at the start of a disk line, on the other hand, allows the data offset for each block to be an even multiple of the block size. When a file is migrated to tertiary storage, its intrinsic metadata must be rejoined with the rest of the file. This presents little additional delay, however, especially when the multi-second access latency to tertiary storage is taken into account. The difficulty of locating the appropriate metadata within the disk line is solved by encoding its position in the block descriptor for the first block of the file it describes. A metadata free map is kept in the line descriptor to speed allocation of new metadata slots.

However, the problem of reclaiming intrinsic metadata storage from migrated files must still be addressed. Since intrinsic metadata is relatively small — fewer than 64 bytes per file — this problem can be solved by

infrequent garbage collection of unused metadata slots. Section 3.3.3 shows that the MSS at NCAR transfers fewer than 500 files per hour, or 12,000 files per day. Even if the rate for a future storage system is 10 times higher, unused metadata slots will occupy less than 8 MB of disk if they are purged daily. The purge can easily be done as part of the migration process described in Section 4.2.3.

4.2. File System Operation

This section describes the actual operation of the RAMA file system, using the design description from previous sections. It describes how data is accessed, specifically detailing the sequence of operations for reads and writes. The methods for free space allocation and other disk management issues are covered in this section, as is RAMA's integration with tertiary storage.

4.2.1. Access to File Blocks on Disk

To make either a read or a write request for a file in RAMA, the client node first hashes the (*bitfile ID, block number*) pair to compute the disk line in which the data is stored. The request is then sent to that node. The client node does nothing further until the request completes and the data is ready. Once the server node with the data receives the request, it reads in the line descriptor for the disk line with the desired data if the descriptor is not yet cached in memory. At this point, the paths for reads and writes diverge.

For a file read, as diagrammed in Figure 4-5, the line descriptor is searched linearly for an entry that matches the desired bitfile ID and block number. If it is found, the block is read from disk and returned to the requesting node. If it is not found, a message is sent to the tertiary storage manager (see Section 4.2.3) requesting the block. When the block has been retrieved from tertiary storage, it is returned to the client. It may optionally be written to disk as well. This approach to file reads works very well if all block reads to the file system correspond to real data. File blocks on disk are retrieved with minimal overhead. Blocks on tertiary storage require more software overhead, but that overhead is lost in the long latencies required to access tertiary storage. If a file block simply does not exist — perhaps because a program requested an erroneous bitfile ID — this scheme will not be able to detect the error rapidly. Such an occurrence is unlikely, however, since bitfile IDs can only be obtained by looking in a directory and directories only contain valid bitfile IDs.

If the operation is a write, shown in Figure 4-6, the server node still looks up the information in the line descriptor. If the block already exists, the new data overwrites the old data. If, on the other hand, the data does not already exist on disk, new blocks from the appropriate disk line are allocated. Blocks marked as free are used first. If none are available, blocks marked clean are used in order of "desirability." The migration manager bases this value on a file's age, size, last reference time, and other factors using a file migration algorithm, and sets it when a file block is written to tertiary storage. If all of the blocks in a line are dirty, RAMA sends an emergency message to the migration manager, requesting that the line be cleaned. The request cannot finish until this is done. This last resort is similar to thrashing in a CPU cache, and exhibits poor performance. After the data is written, the line descriptor is updated to reflect the new access time and mark the written blocks dirty.

This description makes no mention of keeping the file system integral and consistent. Since valid file blocks may be overwritten with other valid file blocks, file system integrity is a major issue. Section 4.3.3 discusses several options for keeping the file system on disk consistent.

4.2.2. Disk Storage Management

RAMA must perform several disk storage management functions. First, RAMA must insure that sufficient space exists to write new files. Some space is freed by applications and users deleting files. RAMA must also migrate data from disk to tertiary storage both to insure that a safe copy exists on tape and to make

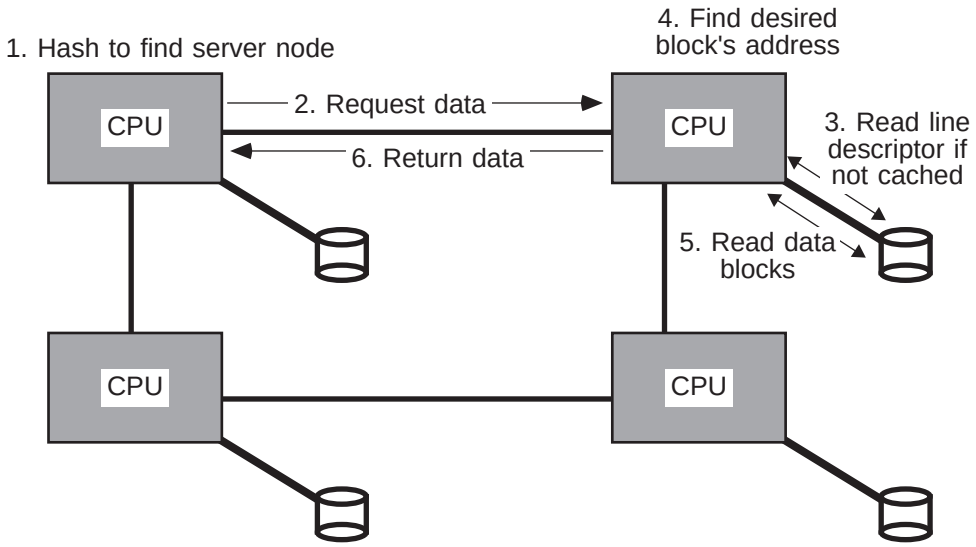

Figure 4-5. A file read in RAMA.

This diagram shows the sequence of operations executed by RAMA for a file read. Note that only the node requesting the data and the node serving the data are involved in the request. This allows RAMA to scale well with additional nodes, as each request involves the minimum two nodes necessary to satisfy the request. There is no mediating node to cause a bottleneck.

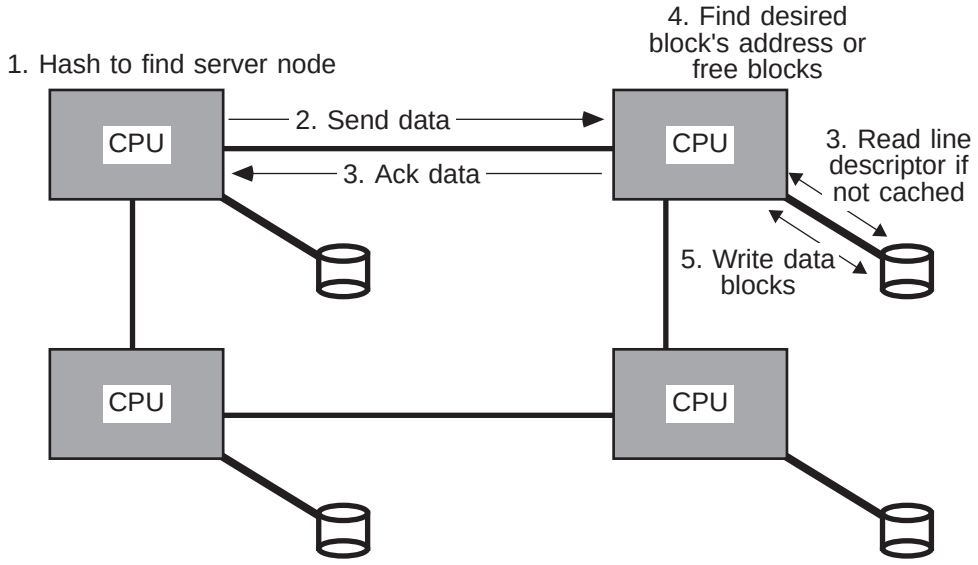

Figure 4-6. A file write in RAMA.

This figure shows the sequence of operations RAMA performs to complete a file write. Like the read shown in Figure 4-6, writes only involve two nodes — the node writing the data and the node with the disk the data is stored on.

additional blocks available for new file storage. File migration design alternatives in RAMA is described in

the next section.

RAMA also attempts to reorganize storage to improve access efficiency. As Section 4.1.2.1 mentions, sequential file data might be written to non-sequential blocks in a disk line. This could occur if the disk line being written to did not have enough contiguous free space to hold all of the sequential file data, as depicted in Figure 4-7. In this case, the data would be written to the disk line in whatever free space was available. The file write could complete promptly, but it would take more time than if the data were stored sequentially.

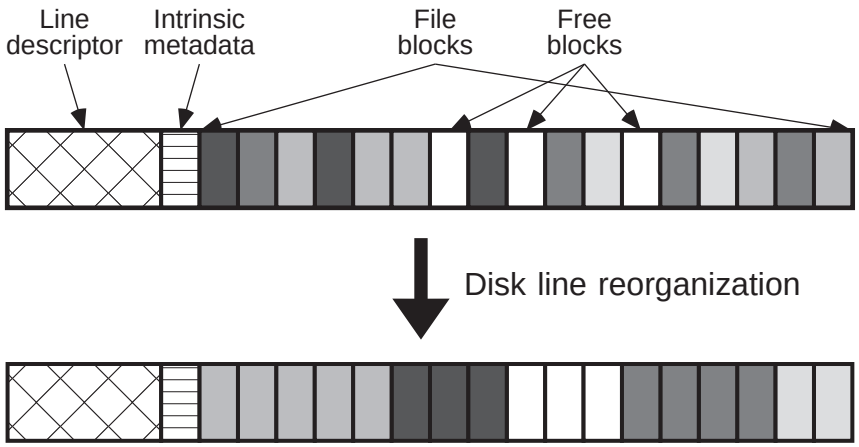

Figure 4-7. Disk line reorganization.

When data is written in RAMA, some consecutive file blocks are stored in the same disk line to allow larger sequential reads. However, there may not be enough contiguous free blocks to write all of the file blocks sequentially. Disk line reorganization solves this problem by switching the positions of blocks within the disk line to enable longer sequential reads and writes. Since the CPU controlling the disk is the only processor that knows the exact physical location of file blocks on its disk, this reorganization can complete without notifying other nodes in the system. In addition, reorganization is not a necessary process. It is merely a performance optimization to create longer sequential transfers, and does not cause a net allocation or release of disk blocks.

Reorganizing disk lines is similar to the LFS [74] task of compacting disk segments. However, RAMA can still run even if disk lines are never reordered because reordering never produces additional free space and is only used to improve performance. Additionally, reorganization is only necessary if a disk line becomes fragmented, making it impossible to allocate several consecutive free blocks. In RAMA, an individual node can rearrange the disk lines on its disk. Only the server node for a file block knows the exact location of that file block, since other nodes in the RAMA system know only what disk line contains the data. As a result, a node can reorder the blocks in its disk lines without notifying other nodes in the file system. The reorganization, pictured in Figure 4-7, allows RAMA to cluster sequential file blocks together by switching their positions in the disk line. One method for doing this is to read the entire disk line into memory, logically switch some file blocks' positions to ensure sequentiality, and write out the line descriptor and blocks whose positions have changed. This can also be done piecemeal, by swapping just a few blocks at any time. In a scientific computing environment, however, free space in disk lines will likely remain unfermented. Many files in a supercomputer environment are large and thus allocate their full complement of consecutive blocks in each disk line. When such a file is deleted or migrated, it releases those consecutive blocks, making them

available for another large file. As a result, fragmentation is only an issue if the file system contains few large files; however, performance on large files is not as important if there are few of them.

4.2.3. Tertiary Storage and Rama

RAMA is designed to be used in high-performance computing environments requiring many terabytes of storage. File migration to and from slower, cheaper media must be well integrated into the file system. RAMA's data layout on disk is designed to facilitate such migration.

Tertiary storage is integrated into RAMA via one or more user-level storage managers. Whenever a block of data is not found on disk, a tertiary storage manager is queried to find the data. Clearly, this method introduces longer latency than a kernel-based storage manager would. However, latency to tertiary storage is already well over a second; the additional few milliseconds make little difference in overall request latency. It is likely that RAMA would use prefetching as well as request batching, since disk file blocks are only 8 KB to 32 KB long, while tertiary storage blocks might be as long as several megabytes or more. In such a case, RAMA might fetch all or much of a file from tertiary when a single block from the file was requested. If this strategy was not optimal — for example, a user might scan the first thousand bytes of each of one hundred one gigabyte files — RAMA would not have to read the entire file. Managing tertiary storage at user level also allows the use of different storage managers, permitting the integration of new devices and new algorithms for managing file migration without recompiling the operating system kernel.

Migration from secondary to tertiary storage is also managed by user-level processes. There may be more than one of these processes, but they will likely be coordinated to avoid duplication of effort. This is not a requirement, however. These processes, called *migration managers*, direct the copying of files from secondary to tertiary storage. RAMA has special hooks into the file system to allow this, though they are only available to programs run by the superuser. Migration managers are allowed to change the state of a file block, marking dirty blocks as clean. They may also adjust the modification time of a clean block so it will be more or less likely to be written over as more disk space is needed. However, migration managers use the standard file system interface to actually transfer file data between disk and tertiary storage.

A typical migration manager searches through every disk line looking for dirty file blocks older than a certain time. This finds file identifiers that are good candidates for migration to a lower level of the hierarchy. This task is easily parallelizable, using one migration manager for each disk. Each process reads and scans all of the line descriptors on a single disk. This is not a long procedure; a 1 GB disk has less than 4 MB of line descriptors which may be read and scanned in a few seconds. The results from all of these processes are reported to a high-level migration manager. This migration manager decides which files will be sent to tertiary storage, and manages their layout on tertiary media. It also optimizes scheduling for the tertiary media readers, trying to minimize the number of media switches.

Once a file has been written to tertiary storage, its blocks become available for reuse. However, these disk blocks are not immediately freed; instead, they are marked as clean so they may be reclaimed if necessary. There is usually no reason to free blocks once they are safely stored on tertiary media, as they might satisfy a future file request. However, the blocks' modification time might be changed. The migration manager could, for example, decide to preferentially keep blocks from small files on disk. If so, it would mark clean file blocks from large files as being older than blocks of the same age from small files. This will not confuse the file system, as a whole file's modification date remains unchanged, as does the modification date for dirty blocks. Only clean blocks which need not be written to tertiary storage may have their last access dates changed.

This architecture fits well into the Mass Storage Systems Reference Model [13]. RAMA itself acts as a bitfile server and storage server for magnetic disk. The user-level tertiary storage managers are bitfile servers for tertiary storage devices; however, they do not necessarily act as storage servers for these devices.

4.3. Implementation Issues

4.3.1. Hashing Algorithm

As [3] and [24] noted, parallel file systems without bottlenecks can achieve near-linear speedup on scientific workloads if data is distributed evenly among all disked nodes. A file system for scientific computation introduces additional problems, however. The first is sequential access. Most scientific computation involves large sequential reads and writes to the I/O system [59], so these requests must run quickly. If each 8 KB block resides on a different processor-disk pair, a single half-megabyte read would need to contact 64 different nodes. Since there is a per-request overhead for each disk, this approach is inefficient. Also, it may excessively congest the interconnection network, as a single I/O sends messages to many different nodes. The disks are also being used inefficiently if many processes are using the file system: instead of few large requests, disks see many small requests and spend their time seeking to the correct locations. To address this problem, the hashing algorithm is designed to keep sequential blocks from the same file in the same disk line.

The hashing algorithm thus must spread unrelated blocks across disk while preserving small subsets of sequential blocks within a file. RAMA requires that a hash function *h* map a 64-bit and 32-bit integer to a random positive number. The disk line a particular file block is assigned to is determined by Equation 4-1:

Equation 4-1. $$diskLine = h(bitfileID, \left\lfloor \frac{blockOffset}{sequentiality} \right\rfloor) \text{ MOD } totalLines$$

Sequentiality is the parameter that determines how many consecutive blocks from a single file will go to a single disk line. While a single run of *sequentiality* blocks will go to one disk line, different runs of consecutive blocks from a file will likely go to different disk lines. Since the hash function is pseudo-random, though, this distribution is probable but not certain. Chapter 7 discusses the performance implications of using a hash function to distribute data.

4.3.2. Interconnection Network Congestion

A basic assumption for the RAMA design is that the interconnection network will not be the bottleneck for the file system. This is a valid assumption because a today's interconnection networks run at 100 MB per second, while small inexpensive disks have sustained transfer rates of under 10 MB per second. However, even high bandwidth networks can become congested. If both requests and data to fulfill those requests are evenly spread around the parallel processor, congestion will not be a constant problem. As Section 6.1.1 will show, RAMA performs even with bandwidths as low as 15 MB/s between nodes of an MPP. Moreover, Sections 7.3.1 and 7.3.2 will show that temporary "hot spots" in the interconnection network are more common in a striped file system than in RAMA.

4.3.3. Data integrity and availability

As with any file system, data integrity is a major issue. The problem is especially acute in RAMA, since a single file may be spread over many disks run by many different processors. Similarly, data availability becomes a problem when parts of a single file are stored in many different places, as the file is unavailable if any of those disks or processors is down.

Data integrity is the more important issue, as a file system must never lose data entrusted to it. Additionally, the file system must insure that a block is not "owned" by the wrong file, as doing so could allow data to be accessed by someone who does not have the proper permission. In addition, a file system must remain consistent, insuring that a crash at an inopportune moment will not corrupt the file system's data structures. After a crash, the file system must insure that every block on disk belongs to exactly one file, or is free.

Since RAMA is designed for MPPs running scientific workloads, it is not so crucial that an application know exactly when a write is complete on disk. Many long-running programs make checkpoints of their state [69] so they can just use the last complete checkpoint if a crash occurs. So long as a crash does not corrupt existing files, a program can restart using the most recent complete checkpoint. In a scientific computing environment, losing the last few seconds of file I/O is not fatal if the application is notified of the loss, since the data may be regenerated by rerunning the all or part of the application.

One option is to use self-identifying blocks on disk. Each block would reserve a few words to record the file identifier and block number that the data in the block corresponds to. This method has several significant advantages. First, crashes no longer present any problem since the line descriptor can be rebuilt by scanning the disk line. Each node can rebuild the line descriptors on its own disks independently by reading each disk line, reassembling its line descriptor, and writing the descriptor back. Since the process uses large sequential disk reads and writes, rebuilding all of the line descriptors on a disk can be done in little more than the time necessary to read a disk at full speed — about 330 seconds for a 1 GB disk that can sustain a 3 MB per second transfer rate. To avoid even this small penalty, the file system assumes that all descriptors are correct, and only rebuilds one when it finds a disagreement between a line descriptor and the self-identification for a block in its line. Another advantage for this method is that line descriptors may be written back lazily. This represents a trade off between faster crash recovery time after a crash and improved performance during normal operation. All of these benefits are countered by a few drawbacks, however. One problem is the increased amount of metadata the file system will need. The overhead for metadata would double with a naive implementation that keeps a copy of all metadata in the file block as well. Keeping a full copy is unnecessary, though, and this overhead is only an additional 0.2% in any case. More importantly, though, a file block is no longer the same size as a disk block, and file blocks are no longer a power of two bytes long. Many programs are optimized to take advantage of specific file block sizes, and it is likely that this choice would cause poor performance.

A better option for maintaining consistency is to introduce a fourth state — reclaimable — for file blocks. This state would explicitly mark those clean blocks that may be reclaimed, and include a timestamp indicating when they were so marked. Such blocks contain valid data for which copies exist on tertiary storage. However, if a crash has occurred more recently than when the blocks were marked as reclaimable, the blocks are considered free. The fourth state thus allows RAMA to use all of the available disk space as a cache for tertiary storage while still keeping sufficient reallocatable space.

Under this scheme, shown in Figure 4-8, all file data is written out before the line descriptor is updated. Clearly, this presents no difficulties if both updates are completed without a system crash. The only difficulty, then, is if the system crashes between the time that the data is written to disk and the time the line descriptor is updated. If a file's blocks are overwritten by new data for the same blocks, the line descriptor still contains the correct description for the blocks. If free blocks are overwritten and a crash occurs before the line descriptor is updated, the blocks are not part of any file and are still marked free. Again, the last few seconds of data are lost, but the file system remains consistent. However, if reclaimable blocks are reused for different files, the line descriptor will still think they belong to the original files. The file system then uses the rule that reclaimable blocks are invalid if a crash has occurred since the blocks were marked reclaimable. The new data written to the blocks is lost, but the file system remains consistent as the blocks are now marked free. Since the blocks had to be clean before they could be marked reclaimable, any data in them can be retrieved from tertiary storage.

Since RAMA does not keep separate free block free lists as used in other file systems, blocks on disk cannot be "lost." Each block's state is listed in its block descriptor, so RAMA can quickly rebuild its per-line free block maps after a crash. Additionally, RAMA never needs a file system-wide consistency checking program. This is a necessary criterion for a file system that integrates tertiary storage, since checking a multi-terabyte archive could take days.

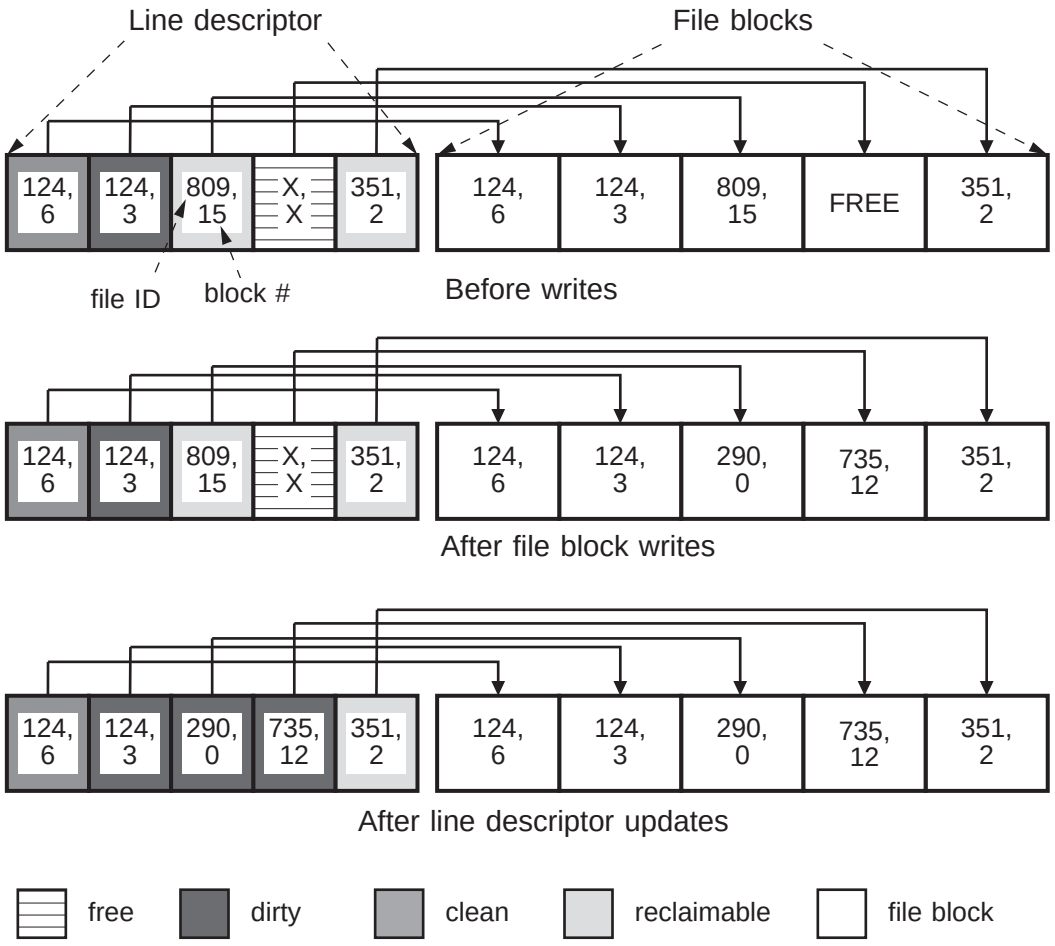

Figure 4-8. Scheme for insuring consistency after a RAMA crash.

RAMA, like all file systems, must have a way of insuring file system consistency after a system crash. This is particularly difficult in RAMA, since clean blocks allocated to a file may be reallocated to a different file without updating metadata on disk. RAMA avoids the problem by introducing a fourth state a file block can be in — reclaimable. Any block in this state is valid only if the block was marked reclaimable since the last crash. Otherwise, the block is free. In this way, RAMA can write data blocks first and then update the line descriptor.

In this diagram, the disk line starts with the state at the top. Next, the file blocks are written, producing the state in the middle disk line. Note that two block descriptors in the line descriptor are incorrect. If a crash were to occur before the line descriptor was updated, however, each node would scan the line descriptors on its own disks in parallel. In this example, the two overwritten blocks and the single reclaimable block would all be marked as free. Recent data would be lost but the file system would remain consistent.

If, as expected, the system does not crash, the final result will be the bottom disk line. In it, the block descriptors all point to the proper file blocks and the modified blocks are marked as dirty.

File availability is another problem that RAMA must address. Uniprocessor file systems spanning more than one disk may arrange disks in a RAID [9] to keep data available even when a disk has failed. It should be possible to use similar techniques for RAMA. However, it is not clear how they would be integrated into the file system, since each node may rearrange its own disks without notifying other nodes. RAMA can uti-

lize techniques learned from RAID to provide availability even when a disk or processor fails. The precise method for accomplishing this is not addressed by this thesis, but remains open for further research.

4.3.4. Disk Storage Utilization

Since file blocks in RAMA are assigned to disk lines by a pseudo-random algorithm, some line in the file system will fill up with valid data while others have plenty of free space. RAMA's use of tertiary storage, can mitigate the problem by using migration to balance the storage load. Unlike traditional file systems that must keep 10-20% of the disk blocks free [54,73], RAMA can fill the entire disk with valid data so long as there are enough clean blocks to reallocate to new files.

A disk line in RAMA is considered full only if all of its blocks are dirty. Free blocks may be reused immediately, and clean blocks can be converted to reclaimable blocks and then reallocated without referencing tertiary storage. If most or all of a line's file blocks are dirty, they must be quickly copied to some other location in the storage hierarchy so future writes can proceed at full speed. This can be done quickly if the MPP running RAMA has one or more relatively large disks running a conventional file system attached to it. This additional storage is considered part of the tertiary storage system, but it has much lower latency and higher bandwidth than tapes or optical disks. Data that must be moved here can be retrieved with latencies on the order of 100 ms or less, as compared to the multiple second penalties that tape drives impose. Since the external storage will be used infrequently — ideally for fewer than 1% of total accesses — its lower performance will have little impact on overall system performance. In this way, RAMA can greatly improve the disk storage utilization of the disks within the MPP at little performance and hardware cost.

4.4. Conclusions

This chapter describes the design of the RAMA file system, which uses pseudo-random distribution to get good file system performance from an MPP with disks attached to each processor node. Since every file operation in RAMA involves only the node making the request and the node storing the requested data, the RAMA design scales well to hundreds of processors. In addition, RAMA is well-integrated with tertiary storage, as it provides transparent file block migration. Rather than include complexity in the disk file system, RAMA keeps access to disk as simple as possible. The complexity is left to the tertiary storage system, where software latency pales in comparison to the multi-second latencies of tertiary storage devices.

RAMA uses the *disk line* as a basic container for file blocks on disk. Each disk line contains hundreds of file data blocks and a *line descriptor* — metadata describing those blocks. A file block is mapped to a disk line by hashing the bitfile ID and block number of the block. Within each disk line, however, file block placement is managed only by the node controlling the disk storing the disk line. Rather than keep a single table of all the blocks in a single file, RAMA relies on the combination of hashing and the disk line table of contents to locate a file block. As a result, any node in the MPP can find the disk on which any piece of data is stored without the assistance of a central directory. Each block in the file system is pointed at by exactly one file system pointer — an entry in the appropriate line descriptor. This scheme keeps RAMA consistent, as a block is always either free or allocated to the file as described in the block's entry in the line descriptor. Additionally, the file system sequences file block writes and line descriptor updates so that a file can only contain data written to it, and no other data. This relationship is preserved even if a line descriptor update is delayed, allowing better performance by postponing metadata updates.

The remainder of this thesis describes simulations to explore the RAMA design space and the design's sensitivity to advances in technology. RAMA's performance is also compared to that of simulated striped file systems, showing that the RAMA design performs comparably to conventional striping. In the following chapters, we will demonstrate that the combination of the RAMA design's performance, scalability, simplicity, and integration with tertiary storage make it a good file system choice for the massively parallel processors of the future.

5 Simulation Methodology

When RAMA was designed, there was no hardware platform available to run such a file system, as it required a massively parallel processor with one disk per processor and relatively high-speed node interconnects. Instead, the concepts behind the file system were tested in a simulator. This method has several advantages over building a real system: implementation speed and the ability to test design decisions over a wide range of applications and configurations.

This chapter first describes the simulator that models the RAMA file system. The event-driven simulator includes parameterized models of disks and MPP interconnection. It has the ability to "run" applications to generate workloads for the file systems. The simulator does not actually execute every CPU instruction in the MPP applications. Instead, it uses program skeletons to model applications as sequences of file system requests separated by delays based on the real applications' access patterns. In addition, the simulator uses several parameterized synthetic workloads to stress the file system design.

The program skeletons we have used to drive the RAMA design are abstracted from real applications running on MPPs. The selected applications are all I/O-intensive; programs that do little I/O will be handled equally well by most file systems. The real applications whose skeletons are used in the studies are based on dense matrix decomposition and global climate model codes. LU matrix decomposition [33,34] is used to solve large systems of simultaneous linear equations in fields such as airplane design. These equations can involve more than one hundred thousand equations in as many independent variables, making the resulting matrix too large to fit into memory. The global climate model code simulates the behavior of chemical species in the atmosphere capturing their interaction with standard components such as moisture, CO_2 concentration, and temperature. Since the chemical model is loosely coupled to the standard radiation and convection model, the file system is used to transfer data between the two. A global climate model, whether standard or chemical, writes its state out every 12 or 24 simulated hours, since scientists do not need more frequent samples for their studies. However, a model actually computes new state every 5 to 30 simulated minutes, depending on the model. Because the standard model passes its state to the chemical model every iteration rather than every 12 hours, it requires two orders of magnitude higher write bandwidth.

The remainder of the workloads are synthetic. One set represents the large fraction of MPP applications with regular data access patterns. The generator for this workload produces a stream of references derived from parameters such as access size, distance in the file between accesses, and computation necessary between accesses. Another workload generator models small file references on an MPP by making many simultaneous small references to randomly selected files.

5.1. Simulator Design

The RAMA simulator is an event-driven simulation of a multiprocessor and its disks. Each node in the multiprocessor may have one or more disks, and is connected to other nodes by one or more network links. In addition, each node has a single CPU that may be "used" by simulated applications. The simulator was designed to run on a uniprocessor; as a result, we only simulated disk I/O and associated network traffic. Simulating the entire computation, including non-I/O network traffic and computation, would have proven intractable. For example, LU-decomposition of a matrix with 16,384 elements on a side would require a full day even for a single 50 MFLOP processor. The workstations used for simulation were no faster than this, and many were considerably slower. Additionally, many simulated problems would have taken more CPU time, and so would have been impossible for a workstation to complete in a reasonable time.

5.1.1. Simulator Implementation

The simulator was written in C++ and used the AWESIME threading package [37] with a few enhancements. A program using AWESIME runs as a single process from the operating system's point of view, but may have many threads, each with its own stack and program counter. For event-driven simulation, AWESIME also maintains a "current time" for each thread. A thread must advance its time explicitly; in doing so, it gives up the simulating CPU and allows another thread to be run. As a result, threads may *only* be preempted at points at which a time advancement could occur, such as using a resource or waiting for another event to occur. This restriction greatly simplifies program design by making multiple thread access to shared data structures easier. Threads need not worry that they might be preempted at a random location, allowing them to use simpler non-atomic methods for updating shared data structures.

AWESIME also provides *facilities* which are equivalent to servers in queueing theory. Threads may use facilities to request and utilize a resource, such as a CPU or a disk. Facilities may accommodate one or more threads simultaneously (defined by the program), and maintain a first-come, first-served queue of requests. Threads are suspended while they are waiting for a facility to become available. When a thread does acquire a facility, it may use it for any period of simulated time. However, the real time needed to model this access is independent of the simulated time. The important figure for simulating a multiprocessor file system, then, is not the total simulated time, but the number of events such as disk requests and simulated "context switches." Facilities thus allow the simulator to manage limited-use resources effectively without taking much real CPU time.

The RAMA simulator has three types of resources — disks, network links, and node CPUs. Each resource instance can accommodate a single user at a time, as it makes little sense for a single node interconnection or disk to be used by two processes simultaneously. Instead, these resources are serially shared among threads requesting them. Figure 5-1 shows the interconnections between resources and threads in the simulator.

5.1.2. Disk Model

The simulator uses a relatively simple model to represent the behavior of each disk. The input parameters to the model are listed in Table 5-1. The model does not include disk track buffers, and uses a generic seek time curve based on minimum, average, and maximum seek times. In addition, tracks are assumed to be a constant size regardless of their location on disk. These approximations limit the accuracy of the simulation. However, the approximations affect both the RAMA file system results and those for a striped file system used as a baseline. Since the important results are in the comparison between the two file systems, the lower level of detail is not as significant.

Figure 5-2 shows the sequence of operations simulated for each disk access. First, the request is put into the disk's request queue. Accesses are scheduled using the CSCAN algorithm, which moves the disk arm

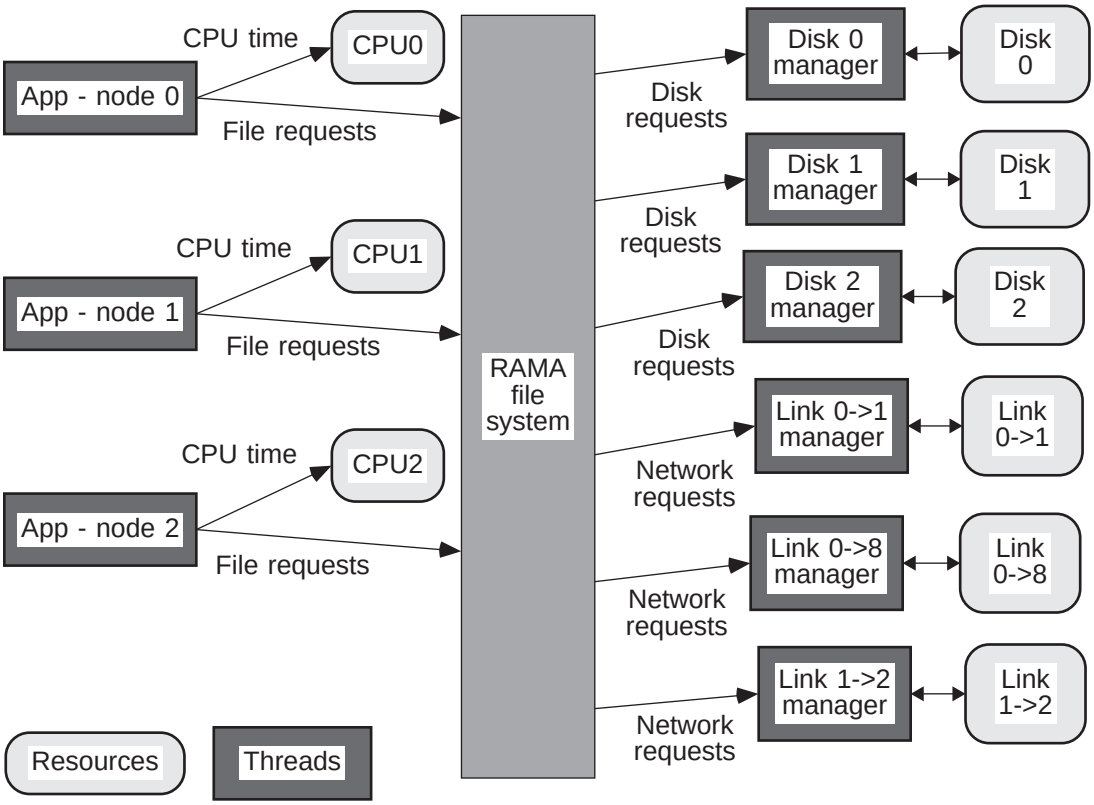

Figure 5-1. Threads and resources in the RAMA simulator.

The RAMA simulator uses threads for two purposes. First, a thread represents an application process on a single MPP node. Second, resources such as disks and network links use threads to schedule incoming requests and manage the devices they model.

steadily from one edge of the disk to the other. At any time, the next request to be serviced is the one closest to the disk arm and ahead of it. Once a request is pulled off the queue, the disk arm must seek to the first cylinder in the request. The simulator calculates the seek time using the distance in cylinders between the two points and the inputs to the disk model in Equation 5-1 [51]. This equation models seek time as the sum of head acceleration, constant velocity travel, deceleration, and settling time. The resulting seek curve for the ST31200N is shown in Figure 5-3. At the same time as the seek is occurring, the active head is switched so it can read or write the appropriate surface. Then, the disk must rotate to the first sector of the request. The simulator keeps track of the rotational position of each disk, and knows where each sector starts. When the head is finally at the correct block, data transfer begins.

5.1.3. Network Model

Impact on the interconnection network is another important factor in the performance of the RAMA file system. To measure this effect, the simulator includes a model of the links between the nodes in a multiprocessor. This model is primarily concerned with the bandwidth used by the file system. The simulator supports several network topologies, including a mesh (Figure 5-4) and three arrangements that assume an infinite-speed “hub” but a finite-speed connection to each node (Figure 5-5). Other simulation parameters affecting

Equation 5-1.

$$A = \frac{-10 minseekTime + 15 avgSeekTime + 5 maxSeekTime}{3\sqrt{cylindersPerDisk}}$$

$$B = \frac{7 minSeekTime - 15 avgSeekTime + 8 maxSeekTime}{3\sqrt{cylindersPerDisk}}$$

$$seekTime(dist) = \begin{cases} 0 & \text{if } x = 0 \\ A\sqrt{dist - 1} + B(dist - 1) + minSeekTime & \text{if } x > 0 \end{cases}$$

Parameter	Value for ST31200N disk
Disk surfaces	9
Cylinders per disk	2700
Bytes per track	40 KB
Disk capacity	1 GB
Revolutions per minute	5400
Minimum seek time	1.0 ms
Average seek time	9.0 ms
Maximum seek time	22.0 ms
Internal transfer rate (ZBR)	27 to 45 Mbits/s
External transfer rate (peak)	4 MB/sec synchronous 10 MB/sec asynchronous

Table 5-1. Parameters for the simulator's disk model.

This table lists the parameters describing a disk modeled by the RAMA simulator. The second column gives the relevant values for a Seagate ST31200N [78], a 3.5" low-profile drive with a fast SCSI-2 interface. This drive formed the basis for the models used in simulation, as it was the 3.5" drive with the highest capacity in the Seagate line at the time.

the network are summarized in Table 5-2. The effect of network configuration on RAMA's performance is studied in Section 6.1.1.

The simulator models the interconnection between the CPUs in the multiprocessor as a set of resources. Each processor has one or more links to the other processors in the MPP. These links have a finite bandwidth and only allow a single message at any given time. Thus, this model will experience network congestion if too much data is sent over a single link.

The model also includes a delay parameter for each packet. During this time, the node sending the packet executes any software necessary to get the data to its destination. This delay is incurred once per packet. In the simulator, packets are large — up to 32 KB by default. This is done to cut down on time spent simulating the interconnection network. It takes approximately the same time to model the delivery of a single packet as it does to simulate a single disk I/O. If packet sizes were very small, most of the simulator's time would be spent modeling the network. Large packets actually make network latencies worse than small packets. A single 32 KB packet will use a 100 MB/s link for 320 µs, queueing up other packets behind it. Using many

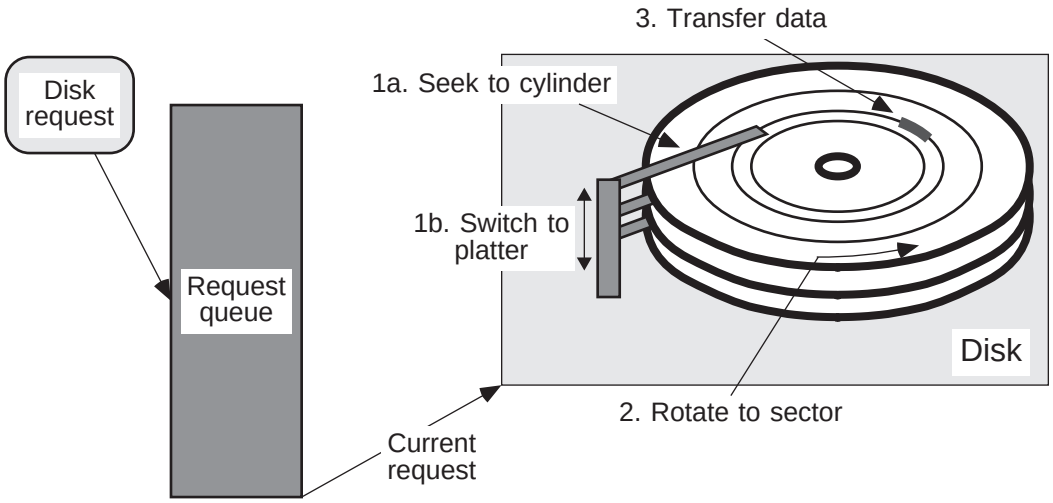

Figure 5-2. Sequence of operations for a single disk request.

This is the sequence of operations that the simulated system models for a single disk access. A disk request is queued, and serviced in CSCAN order. When a request is pulled off the queue, the disk seeks to the correct cylinder, switches heads to the correct track, and rotates to the first block in the request.

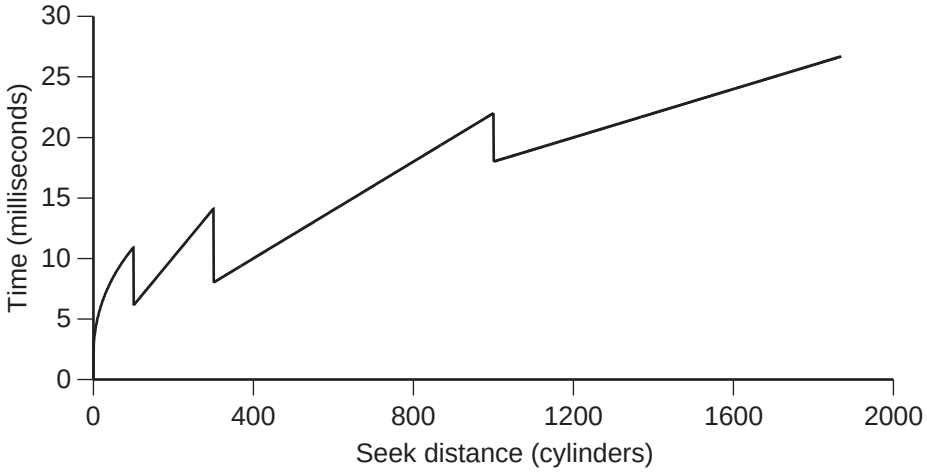

Figure 5-3. Disk seek time curve for ST31200N.

This graph plots seek time against seek distance (in cylinders) used by the simulator to model the Seagate ST31200N. The seek time in this graph is not empirically measured. Instead, it was generated using the formulas in Equation 5-1.

small packets, on the other hand, allows a single small transfer queued behind a large one to complete more quickly. Since control packets for the file system are typically small, large packets could only hurt RAMA's performance.

Parameter	Typical value
Network topology	mesh
Link bandwidth	100 MB/s
Maximum packet size	32 KB
Packet latency	10 µs

Table 5-2. Simulation parameters for the interconnection network.

All of these parameters govern the network model used by the RAMA simulator. The performance figures used are less than those attained by the Cray T3D. The packet size is larger than the normal packet size used in MPPs. This choice was made to speed up the simulation. Large packets actually cause greater network congestion by preventing small packets from getting through quickly. Thus, the results are more pessimistic because of the granularity of network transfers.

While the simulator supports other topologies, most experiments used the mesh topology because it most accurately represents today's MPP interconnection networks. Other topologies, such as the hub, might be more applicable to a network of workstations used as a single parallel computer. Section 6.1.1 examines the effect on RAMA's performance of varying these parameters.

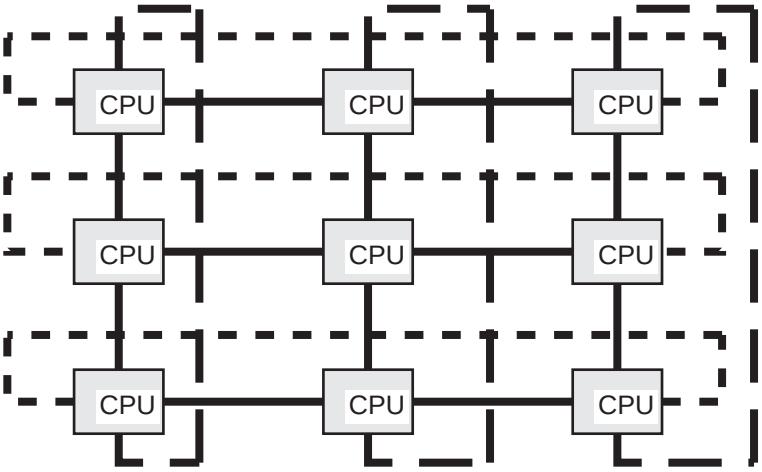

Figure 5-4. Mesh network topology.

The mesh network topology has become more common in modern MPPs because it is considerably cheaper than topologies such as the hypercube. In a mesh, every node has four links — one each to each of its neighbors in a two dimensional plane. Nodes at opposite edges may be connected to each other as indicated by the dashed line. If this is the case, as it was in the simulations, the mesh network is a torus.

The simulator uses a store-and-forward model for any packets that must traverse multiple links. The route is chosen in advance by the simulator, and is independent of any existing congestion. For the mesh network, for example, the simulator sends a packet across and then up or down to its destination. All packets from one specific node to another follow the same path. While these routing algorithms would perform poorly in a heavily-loaded network, the data in Chapter 7 show that the simulated file systems do not cause congestion. Additionally, the actual load on individual network links was very low — less than 3% for 100 MB/s links. The low level of congestion meant that there were few packet "collisions," so the choice of a store-

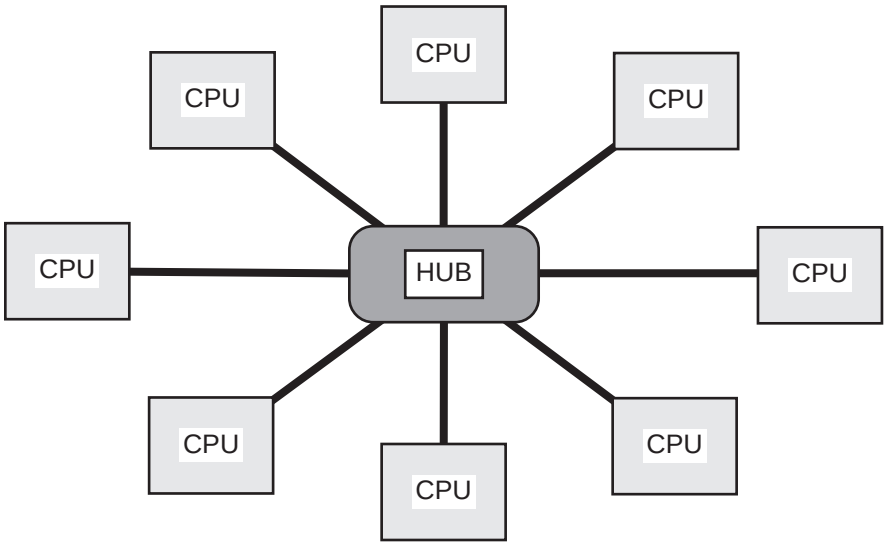

Figure 5-5. Star network topology.

While most experiments used a mesh topology, the simulator also supports simpler connection schemes. In particular, a star network might be used by a network of workstations. In such a network, each computer has a single link, and all links meet in a very high bandwidth switch. This nexus is often implemented using a crossbar switch. A star network is attractive because it uses only one link per node, and all messages must only travel two hops. However, it does not scale well since it is very difficult and expensive to build a hub to support hundreds of high-bandwidth links simultaneously. Many ATM networks use this network topology, also called a *hub-and-spoke* topology.

and-forward model rather than other methods used in MPPs had little effect on network bandwidth and latency.

A packet must be fully received by a node before it can be resent on a different link. However, the latency to send a message is only paid by the processor that first sends the packet. Subsequent routing is assumed to occur in hardware, and does not slow the message down. The simulator chooses the shortest route for each packet. There is no need for a routing algorithm for the network topologies that do not provide multiple paths between nodes.

5.1.4. Multiprocessor CPU Model

The simulator models each processor in the MPP as a single AWESIME facility. Any application program that executes code on a processor must wait until the processor is available to run it. The simulator does not support preemptive multitasking. For many simulations, this presented little problem. The experiments in this thesis involved running just a single application over the entire machine, so there was no alternate program that could use spare (simulated) CPU cycles. Other simulations might model more than one application running on the MPP, since current parallel processors use multiprogramming to run several jobs at the same time. This thesis does not include simulation of multiprogrammed workloads; modeling of multiple simultaneous programs on an MPP is suggested as future work in Chapter 8.

File system requests themselves were not required to wait for CPU time for two reasons. First, including them would have complicated the simulation significantly by requiring preemption to be added to AWESIME. Second, file system code would require very little CPU time on a processor with a single disk, as

RAMA would be configured. A single file I/O request takes approximately 20,000 instructions to execute. This would require 400 microseconds on a 50 MIPS processor, the minimum which was simulated. If a single disk were able to complete 100 I/Os per second, its CPU would use 40 milliseconds — 2.5% of total CPU time. For faster processors, this number would be lower still. This assumes no more than a single disk per processor. In the striped file systems that were simulated, however, a node in the MPP could have 32 or more disks attached to it. In those cases, the simulator assumed that a separate dedicated CPU handled file requests the only load on the processor directly on the interconnection network is the added amount of data sent to and from the attached disks.

5.1.5. File System Simulation

The RAMA simulator does not include a full simulation of the file system. Instead, it only simulates the movement of file data between clients and disks, including any network transfers. The simulator does not model RAMA's handling of metadata, and does not simulate file migration. The movement of files between secondary and tertiary storage is limited by the speed of the tertiary storage device, and would consume only a small fraction of the available file system bandwidth. Thus, including file migration would have little effect on applications' read and write bandwidth to secondary storage. Since the transfer rate between disk and most tertiary storage devices is less than a few megabytes per second, any program requiring a tertiary storage access will be limited by the bandwidth of the tertiary storage device; RAMA would perform no better and no worse than any other file system delivering a file stored on tape.

The file system simulation includes the mapping from bitfile ID and block number to a disk line and the node that owns it. The real file system would then read the line descriptor if it were not already cached and search for the block descriptor for the block's address. The simulator, on the other hand, assumes that the line descriptor is cached in a processor's memory and simply reads and writes the data on disk. Since storing the line descriptors would have required too much memory, the simulator assumes that the requested data is stored contiguously starting at a random location within the disk line. Caching all of the line descriptors for a 1 GB disk would require only 2 MB of memory, as shown in Section 4.1.2.1; thus, it is reasonable to assume that positional metadata is cached. Likewise, the assumption that data will be stored consecutively is justified by Section 4.2.2, which presents a scheme for reorganizing disk lines to keep a file's blocks contiguous.

The RAMA simulator fully simulates the remainder of the actions for read and write requests, as diagrammed in Figure 5-6. A file request is sent from the client node to the server node for the requested block. If the request is a write, the message is large enough to include the data being written. The server node then performs the disk I/O. If the request is a read, a packet of the appropriate size is sent back to the client node. No reply is sent for a write, as the sending node assumes the write completed unless a message to the contrary is sent.

The simulator also models a simple striped file system for the performance comparisons in Chapter 7. As for RAMA, the simulator ignores metadata in the striped file system. The striped file system model takes parameters such as stripe size and the number of disks in a stripe and uses them to map file blocks to disks. However, it does not use a sophisticated scheme to allocate blocks within disks. Instead, it places an entire stripe unit (the amount of consecutive file data on each disk) sequentially starting at a random block on the disk.

Since the simulator does not simulate long-term behavior, it was not necessary to include complex space allocation techniques. Additionally, omitting many potential pitfalls of striping allows a comparison of the two layout schemes without the synchronization complexity many striped file systems impose. As Chapter 7 will show, however, RAMA performs comparably even to a stripped-down striped file system.

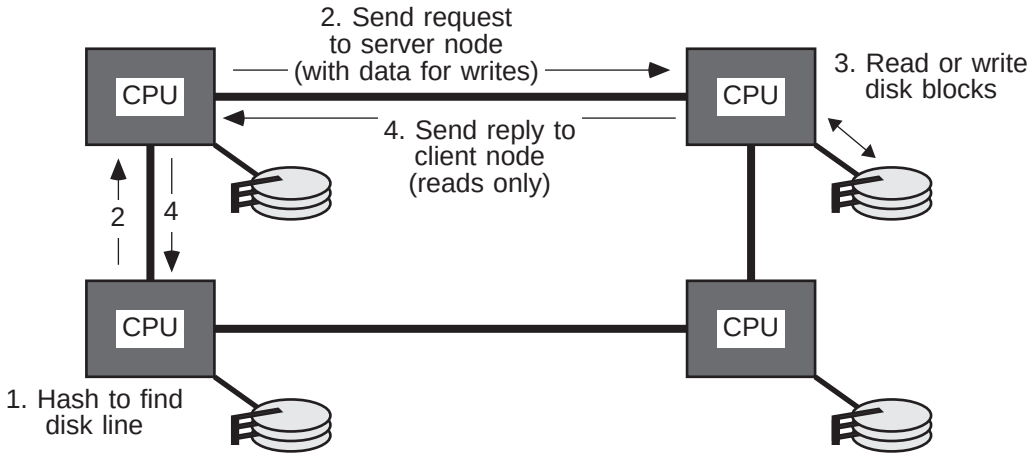

Figure 5-6. Actions simulated for a read or write request.

This diagram shows the actions the RAMA simulator models for a read or write request. First, the simulator determines which disk line and node own the requested data. A message is sent from the client node to the server node with the desired data. If the request is a write, the data to write is also sent at this time. The server node then performs a disk I/O. At this point, a write is complete. For reads, however, a reply including the requested data is sent back to the client node.

5.2. Applications Simulated

As mentioned earlier, the RAMA simulator uses program skeletons from both synthetic and real applications to model the file system accesses on an MPP. Two of the applications — global climate modeling and out-of-core LU decomposition — are abstracted from real programs running on existing MPPs. The global climate model used in the simulator integrates a standard general circulation model with a model of the behavior of chemical species, and is used to predict both future climate and the future concentrations of trace gases in the Earth's atmosphere. LU decomposition solves large systems of simultaneous linear equations. Out-of-core decomposition must be used when the matrix is too large to fit into main memory.

The other workloads in the simulator are synthetic workloads that attempt to mimic common MPP file access patterns. One workload models applications such as seismic data processing that read very large files sequentially and do some processing on each data chunk. Another workload, which is actually a special case of the first, uses all of the nodes in the MPP to read or write a very large file as rapidly as possible. Both of these applications access large files from disk, and both require high bandwidth. A third workload, designed test RAMA's ability to access small files, uses multiple threads on each node to read many small files simultaneously.

Each workload in the simulator was modeled as if it were the only application running on the MPP. The simulator can model multiprogramming; however, this thesis is primarily concerned with showing that the RAMA file system performs well for demanding workloads. Future work will address RAMA's performance under multiprogramming workloads. Since RAMA uses pseudo-random distribution, however, we expect that it will perform well even when two applications "destroy" each other's regular access patterns.

The applications simulated were chosen because they require high file bandwidth and are sensitive to data placement on disk. Many programs that run on MPPs do not request many megabytes per second of I/O. However, such programs also place little stress on the file system. For them, any data layout would be adequate, as file I/O is not their bottleneck. A program that transfers only a few megabytes per second between

disk and memory performs similarly under most modern file systems, including RAMA. Applications that request hundreds of megabytes of file data per second, on the other hand, put a much higher load on the file system. The arrangement of data on disk is critical for these applications, since poorly laid-out data leads to slower file I/O and makes the bottleneck worse. Such programs are becoming more common as faster processors allow users to handle larger data sets which may not fit in main memory.

The application skeletons used in the simulator preserve the I/O request patterns of the original applications as much as possible. However, they replace the computation phases of the programs with delays in the simulator. In this way, a workstation can simulate any amount of uninterrupted computation in a few milliseconds rather than the actual computation time. Simulating programs in this way allows the examination of more design points because each simulation requires hours rather than the days required to actually perform all of the MPP application's computation on a workstation.

For example, decomposing a square matrix with 32,768 elements on a side requires 10^{13} floating point operations (FLOPs). On a 20 MFLOPS SparcStation 2 workstation, these calculations would take 500,000 seconds, or almost 6 days. A typical simulated MPP, however, might have 256 processors each running at 100 MFLOPS, and be able to complete the decomposition in about 400 seconds — hardly a large problem. When the simulator models computation as simply a simulator delay, the model takes only 800 seconds to complete. The speed of the simulation is proportional to the number of I/Os that must be simulated, and is independent of the actual computation time on an MPP. This allows simulation of larger problems, but the difficulty of simulating I/O remains. The simulator needs about 3 ms to model a single 32 KB request from an application, limiting it to 10 MB of I/O per second of simulator time. Applications that have I/O rates over 10 MB per second are thus simulated on a SparcStation 2 more slowly than they would actually execute on a real MPP. This slowdown occurs because a real MPP might run at 25 GFLOPS or more, as compared to 20 MFLOPS for the SparcStation. For some applications with I/O rates of 100 MB per second or more, the slowdown factor was 10 or greater. Since the simulated applications would take no more than a few hours to complete on an MPP, however, the simulation of their file requests took less than a day.

This rest of this section describes the applications modeled by the RAMA simulator. Matrix decomposition and the global climate model are based on real-world applications, while the strided sequential and small file access models are synthetic. Each program's section discusses the program's purpose and describes the program skeleton used to model its accesses. Each program's skeleton is hard-coded into the simulator; however, adding additional skeletons (and thus applications) is a simple matter because the skeletons are written in C++ and compiled with the simulator.

5.2.1. Strided Sequential Access

Many massively parallel applications are characterized by regular accesses to a single file, interleaving computation and I/O. These programs generally fall into one of two categories: readers and writers. Some programs, such as seismic analysis, read a large data file and perform operations on each successive chunk of data. Others, including many simulations, keep all of their data in memory and periodically write out a snapshot of memory. Both of these types of applications, however, exhibit regular access patterns to storage.

Figure 5-7 shows the algorithm that the simulator used to define the access pattern of a program making strided sequential accesses and their effects on the I/O request stream. As the figure shows, one parameter is the time between successive I/Os on a single processor. This quantity is affected by both the amount of computation the application must do, and the speed of a single processor. If problem size and MPP size are held constant, the time is inversely proportional to a single CPU's MFLOP rate. The effects of changing the problem size or the MPP size, however, vary by problem. For problems such as seismic analysis, the amount of computation is proportional to the amount of data processed. Simulations, on the other hand, may simply become more detailed on a faster CPU, increasing computation needs without similarly increasing I/O demands. [58] classifies large applications into two groups: one for which I/O demand increases linearly

```
firstOffset = thisNode * nodeDelta
for i = 1 .. iterations
      offset = firstOffset
      for j = 1 .. chunksPerIteration
            offset = offset + chunkDelta
            Read or write reqSize bytes at offset
      Compute for thinkTime
      firstOffset = firstOffset + iterationDelta
```

Figure 5-7. Algorithm for the strided sequential access application.

The input parameters to the strided sequential access model are *nodeDelta*, *chunksPerIteration*, *iterations*, *reqSize*, and *thinkTime*. *NodeDelta* determines the "distance" in the file between nearby nodes' accesses during an iteration, while *iterationDelta* controls the distance between successive iterations on a single node. Typically, one of these parameters is small and the other large. These two cases are analogous to row-major and column-major accesses to a matrix. *ThinkTime* and *iterations* control the total running time and, in conjunction with *reqSize*, the computation granularity of the application. *ChunkDelta* and *chunksPerIteration* allow the algorithm to model applications that use nested loops to gather data from several parts of the file for a single compute phase.

with processor speed, and another for which I/O demand increases as the 3/4th power of processor speed. Thus, I/O demands grow with processor, though the rate differs between the two classes of programs.

The rest of the parameters to the strided access model determine the order in which the various nodes of the MPP access the file. Request size often depends on the available memory on each node. If an application can use more memory on each node, it will attempt to process more data in each iteration.

The seek distance, or simply distance, between the start of adjacent nodes' I/Os, on the other hand, reflects an application's basic problem decomposition. Some programs prefer to have each node work on its own fragment of the computation. In such codes, the distance between adjacent nodes is large, giving each node its own section of the file. For other applications, however, the distance between adjacent nodes is small. This allows the entire MPP to work as a unit and progress through the file from start to finish. However, each individual node will, under this scheme, perform a long seek between iterations. Thus, the distance between adjacent nodes is generally inversely related to the distance between iterations for a single node.

The remaining parameters are available to fine tune the access pattern or simulate access from nested loops. One allows an application to request several small chunks from a file between each computation phase instead of just one. The other sets the distance between consecutive accesses issued at the same time, as compared to the distance between the starts of requests for successive iterations.

5.2.2. LU Matrix Decomposition

Matrix decomposition is a method used to solve large systems of dense linear equations. For example, aircraft designers use LU matrix decomposition to solve the thousands of equations involved in doing electromagnetic analysis of a stealth airplane. The commonly-used algorithms for decomposition require $O\left(n^3\right)$ FLOPs, where n is the number of elements along one edge of the matrix. As a result, large matrix decompositions are often done on MPPs.

While MPPs solve one difficulty with matrix decomposition, another hurdle remains. The data set for a decomposition is also large, containing n^2 floating-point numbers. For example, a matrix of double-preci-

sion[1] numbers with 32,768 elements on a side requires 8 GB of storage. On a 256 processor MPP, it would require 32 MB of memory per processor just to hold the entire matrix in memory. If each processor ran at 100 MFLOPS, the problem would be solved in 900 seconds, or 15 minutes. Of course, researchers would like to solve much larger systems of equations. [77] reports that scientists would like to solve 150,000 equations in 150,000 variables. This problem would have a 180 GB input matrix and use 2×10^{15} FLOPs. On a 256 processor machine running at 200 MFLOPS per processor, it would take about 12 hours to solve. However, the application would need 720 MB of memory per processor to hold the entire matrix.

Application designers address the storage problem by using out-of-core algorithms. These methods use the disk to store most of the data, and operate on relatively small chunks in memory. The advantage to this method is machine cost — disk is considerably cheaper than RAM. The drawback, though, is that out-of-core algorithms place a high load on the I/O system. For example, decomposing a 150,000 x 150,000 matrix with 256 processors and 32 MB of memory per processor would require approximately 20 TB of I/O. Since the problem would take 40,000 seconds, this is an I/O rate of 500 MB per second.

The simulator implements out-of-core LU decomposition, as described in [93]. LU decomposition transforms a matrix into a product of two new matrices of the same size. One matrix is upper-triangular, and has no non-zero elements below its diagonal. The other result matrix is lower-triangular, and is empty above the diagonal. After the decomposition, back-substitution into the upper-triangular matrix easily yields the solution to the system of equations.

The out-of-core version of LU decomposition divides the matrix into many columns, each of which fits into one quarter of the free memory on a node. Individual columns are divided up statically among processors in such a way that the computational load is as balanced as possible during the entire execution time. The algorithm, shown in Figure 5-8, proceeds left to right, reading in a column, updating it with previous columns, decomposing it, and writing it back to disk. Note that updating a column requires reading the lower-triangular half of every column to its left. As a result, the total amount of I/O done by the algorithm is $O\left(n^4 / M \right)$ bytes, where M is the total size of the memory available across all nodes. Thus, doubling the problem size without changing the machine's characteristics increases the amount of computation by a factor of 8 and the I/O by a factor of 16.

```
for column = 1.. ncols
      Read column from disk
      for prev = 1 .. column-1
            Read lower triangular portion of prev from disk
            Update column with prev
      Decompose column into upper and lower triangular parts
      Write column back to disk
```

Figure 5-8. Out-of-core LU decomposition.

This is the out-of-core LU decomposition algorithm, from [93], used in the simulator. The matrix is divided into columns that fit into memory. Each column is updated with the columns before it, broken into upper- and lower- triangular parts, and saved.

1. A double-precision number requires 8 bytes (64 bits) of storage.

Like many applications on traditional supercomputer applications [60], out-of-core LU decomposition is read-intensive. Each entry in the matrix is written exactly once; for a matrix of size n, n^2 entries are written. On the other hand, a column is read once for each column to its right. While these accesses only read the part of the column corresponding to the lower-triangular portion of the matrix, they are the dominant contribution to the $O\left(n^4 / M \right)$ bytes written. Only when M approaches n^2 do reads cease to dominate. This corresponds to the case that the memory is sufficiently large to hold the entire matrix, making out-of-core solution unnecessary.

Good I/O performance is thus crucial for out-of-core LU decomposition. This is especially true if processors become faster, allowing larger problems, without corresponding decreases in memory cost per megabyte.

5.2.3. Global Climate Modeling

Modeling the earth's climate is another important application typically run on supercomputers and MPPs. General circulation models (GCMs) break the earth into thousands of grid cells, each a few degrees of latitude and longitude on a side. Above each grid cell are ten or more vertical layers, each of which is considered a separate cell in the simulation. The GCM models two types of behavior: intracell physics and convective dynamics. The physics section of the code calculates the effects of phenomena within a cell on the local piece of the atmosphere. These phenomena may (depending on the level of detail in the model) include incoming solar radiation, ocean or ground radiation, and clouds. The processors in an MPP perform little communication during this phase of the code, since each cell may be updated independently of all others. The dynamics portion of the model, on the other hand, tracks the movement of heat and atmospheric compounds (H_2O, CO_2, etc.) between cells and thus requires interprocessor communication. The GCM computes the state of a cell at time $t + \Delta t$ from the state at t in both the cell itself and neighboring cells. For current GCMs, Δt is typically between 5 and 30 minutes of simulated time.

Traditional GCMs do not place a high demand on the I/O system. The equations governing the behavior of the atmosphere are complicated and require a good deal of CPU time. Additionally, climatologists do not need the output of the climate model after every time step. Instead, they take a snapshot of the climate every 12 or 24 simulated hours. These snapshots comprise a "movie" of the climate on Earth over a period of years or decades. A snapshot's size depends on the resolution of the grid the GCM uses. Since current machines are not fast enough to simulate using a very fine grid, they produce less than 50 MB per simulated 12 hours. However, it takes many minutes to simulate 12 hours of Earth's climate, so the overall I/O rate is relatively low — CPU time dominates I/ O time.

Recently, though, there have been some developments that greatly increase GCM demand for file I/O. Researchers now wish to simulate the effects of various chemicals on the atmosphere using a model, called GATOR, that is decoupled from the primary GCM. GATOR is used to simulate these chemical processes in the same way that a standard GCM does: it updates the chemical species in each cell based on the conditions in the cell and its neighbors during the previous timestep. Since GATOR may be run separately from the main GCM, however, it needs to take a snapshot of the GCM's state every timestep, rather than every 12 hours. For a GCM with a 10 minute timestep, this increases the I/O rate by a factor of 72. In addition, the two models prefer different data layouts. The standard GCM lays data out along lines of longitude, while the chemical model (GATOR) groups data in a block-cyclic fashion, as in Figure 5-9. This difference in data layout will cause difficulties for standard striped file systems, but not for RAMA, as Chapter 7 will show.

5.2.4. Small File Access

Since RAMA is designed to support small file access as well as supercomputer applications, one workload creates a stream of relatively short accesses to randomly selected files. The resulting reference stream is

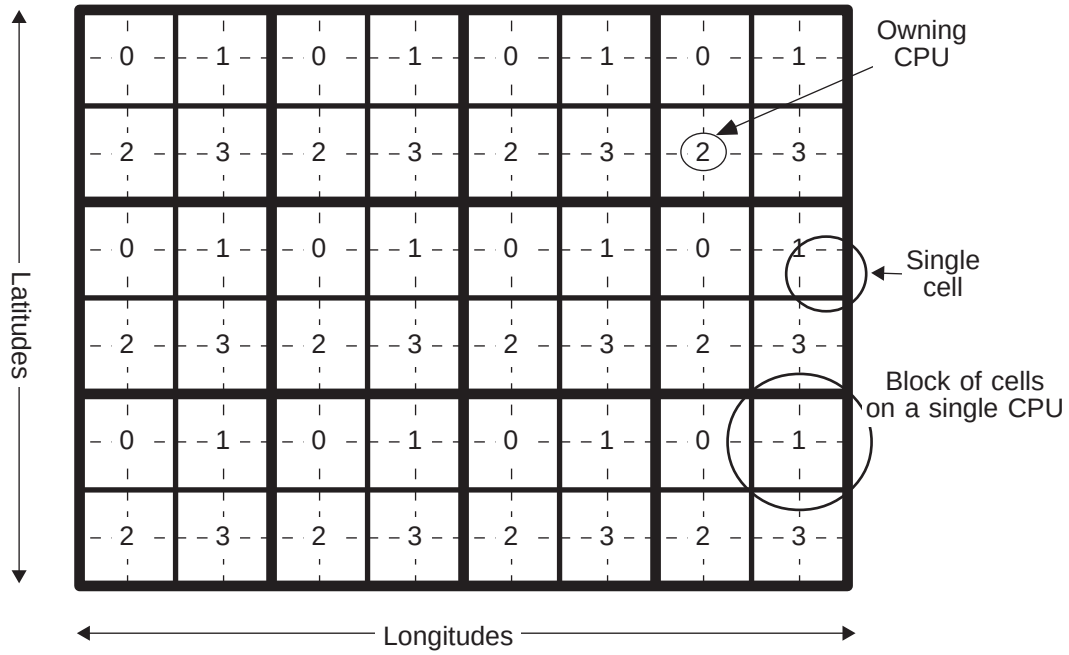

Figure 5-9. Block-cyclic layout of data for GATOR.

GATOR, an application to model the behavior of chemicals in the atmosphere, uses data arranged in a block-cyclic layout. This arrangement provides good load-balancing, as the amount of computation done for a cell varies by the its latitude. Grouping cells on the same processor decreases communications cost. These needs are balanced in a block-cyclic layout. The figure above shows block-cyclic layout for four processors; a real model would use many more processors.

similar to that generated by a network of workstations, as described in [2]. However, this workload is purely synthetic and is not based on actual traces.

The small file access workload generator creates one thread on each MPP node running the “application.” Each thread issues an asynchronous read or write request for an entire small file and then waits a random exponential length of time from the start of the previous request before making its next request. This method of generating requests requires a throttle, though, since it is possible to request files faster than the file system can satisfy requests. Thus, the workload generator includes a parameter that sets the maximum number of outstanding requests for any thread. If a thread reaches that limit, it must wait until a request in progress completes.

The probability that an access will be a read is a parameter to this workload, as is the average delay between the starts of consecutive requests. The bitfile ID of each file is randomly selected, as is the actual interrequest interval. File size, however, is a fixed workload parameter and is thus the same for each file in a simulation run. Since each request transfers the entire file, all requests made by this reference generator are the same size.

This workload does not accurately model workstation file access. Rather, it is intended to gauge RAMA’s ability to simultaneously access many small files. A real network of workstations would generate references to files of widely varying sizes, though real workstation files are small and most references read the entire file [2]. While this workload cannot be used to accurately predict RAMA’s performance as a workstation

network file server, it will show that RAMA performs well while providing many small files. Since RAMA users may use workstations as well as supercomputers, low latency on small file service is essential.

5.3. Conclusions

This chapter describes the simulator and workloads used to test the RAMA file system's key concepts. The MPP hardware and file system are modeled by a multithreaded simulator based on the AWESIME threading package. Disks, network links, and CPUs are all resources managed by the threads in the simulator. Modeling these resources allows the simulator to show RAMA's impact on the MPP. Additionally, each resource's parameters can be altered to predict the file system's performance on future hardware. The simulator uses a simple model of both the RAMA file system and a generic striped file system to validate the basic concepts behind RAMA's design.

File reference streams for the simulator are derived from both real applications and synthetic workloads, all chosen because they place high demands on the file system. The simulator uses program skeletons to generate these reference streams. Each skeleton models an MPP application as a sequence of file requests alternated with computation. By modeling computation as usage of the CPU resource, the simulator can simulate programs that would take days to run on the workstation used for the simulation experiments.

The real applications modeled in detail are LU decomposition and global climate modeling. LU decomposition is used to solve dense systems of linear equations. It is a read-intensive algorithm that makes many passes through a single file as it decomposes the matrix. Global climate modeling, on the other hand, is a write-intensive algorithm that uses the file system to store its output. For the algorithm in the simulator, secondary storage is used as an intermediary between two separate programs cooperating to produce a more accurate atmospheric model. The simulator also uses synthetic workloads to model both generic MPP applications and small file access similar to that generated by a network of workstations.

Chapter 6 will use this simulator to explore the sensitivity of the RAMA design to advances in technology. It will show that RAMA is disk-limited and that pseudo-random data placement does not place a high load on the interconnection network even as disks become faster. Chapter 7 then compares RAMA's performance to that of a striped file system on similar hardware. It will demonstrate that RAMA, without requiring application configuration hints, performs comparably to a well-configured striped file system and far outperforms poorly-configured striping.

6 Sensitivity of the RAMA Design to Changes in Technology, Design and Usage

The expected performance of the RAMA file system design depends on many parameters. These fall into two broad categories: technological parameters and design parameters. A system designer has control over design parameters, but the others are generally determined by the technologies used to construct the MPP. For example, a machine could be built with any number of processors, but the processor speed is fixed by the CPU chips available. In our terminology, the former is a design parameter, while the latter is a technological parameter.

This chapter explores the sensitivity of the RAMA design to various parameters governing the network, disk, memory size, and speed of the underlying processor. These values influence more than just the file system, however. Faster processors allow the solution of larger problems [58], leading to even greater demands on the file system. The goal of this chapter is to understand how well RAMA can scale with advances in technology. The simulation experiments in this chapter project RAMA's performance for various problem sizes as well as for future MPPs.

In this chapter, we will demonstrate that the RAMA design scales well with improvements in technology. Since the file system avoids software bottlenecks during reads and writes, faster disks allow RAMA to provide files at higher bandwidth and with lower access latencies. Network parameters, on the other hand, have little effect on file system performance. Disk latency is much longer than even the 1-2 ms network latencies common today. As a result, varying network latency from 2 ms down to 5 μs only changes overall file system performance by 1%. Similarly, interconnect network capacity affects performance only when the aggregate bandwidth of every link is less than 10 times the maximum bandwidth from the file system. This ratio is satisfied for current systems, and network bandwidth is increasing faster than disk bandwidth.

The graphs in this chapter focus on RAMA's behavior under a special case of the strided access workload — full speed reads. This workload, which reads an entire file without any computation, is used for two reasons. First, RAMA's performance under this workload is similar to that under the other workloads tested, as Chapter 7 will show. Second, the full speed workload is the only one in which an increase in RAMA performance always results in a reduction in program execution time. For most applications, such as matrix multiplication and climate modeling, I/O is overlapped with computation. A program that attains 90% processor utilization with current technology can only gain an additional 10% utilization regardless of how well the file system performs. A file system using future disks that can sustain five times the bandwidth of current disks may be capable of a five-fold increase in file system bandwidth; however, this increased potential bandwidth will not be realized unless the program's CPU time is reduced. While future problem sets will place higher loads on the file system, as Section 6.1.3 will show, we wanted to use the same "problem" while varying the hardware the file system uses. Using a simple abstracted benchmark — full speed reads

— allows the simulations to vary a single parameter rather than changing both the file system parameters and the application parameters.

The first two sections in this chapter analyze the effects of changing parameters on performance. Section 6.1 discusses the effect of technological improvements to disks and networks on RAMA's performance. It will also consider the impact of larger memories and faster processors on bandwidth required by MPP applications. The effects of design parameters on RAMA are then discussed in Section 6.2.

Technological and design parameters are not the only influences on file system performance. Another important factor in a file system's performance is the workload that it must satisfy. Most of the simulations in this chapter involve large file workloads, as MPPs spend most of their time accessing large files. For RAMA to be useful in heterogeneous environments, however, it must also perform well on small file workloads. Section 6.3 will examine RAMA's performance on a workstation-like workload. Simulation results show that RAMA is suitable for small files as well as the large files used by the rest of the workloads simulated in this chapter.

The first three sections in this chapter all show RAMA's expected performance as a single parameter is varied. The disks of the future, however, will have shorter seek times, higher rotation rates, and higher bit densities than today's disks. Similarly, MPP networks and processors will be faster than they are currently. Section 6.4 will consider the combined effects of all of these improvements on RAMA's projected performance.

6.1. Technological Parameters

Technological parameters are determined by the kind of technology used to build the MPP hardware. They change over time, but are generally fixed for a particular machine at a particular point in time. CPU speed, network bandwidth and message latency, and disk bandwidth and latency are all technological parameters.

Modern MPPs are made of workstation-class CPUs connected by high-speed, low-latency networks. Since each individual processor is a commodity CPU used primarily for workstations, per-processor performance increases track those of general purpose computers. According to [40], CPUs increase in speed between 50% and 100% per year, so each node in an MPP doubles in speed in 12 to 20 months.

Interconnection network performance has also increased dramatically in the past several years. The nCube-1 [63], for example, achieved a maximum of 2 MB per second along a single network link. Today's network speeds have increased by two orders of magnitude. The Cray T3D multiprocessor [20] can transfer data at over 150 MB per second between two nodes. Network latencies have also decreased as faster CPUs and better network hardware have cut the software and hardware overheads of sending a message. While the Intel Paragon's message latency is over 150 μs, the latency to send a message on the Cray T3D is less than 2 μs.

Disk performance, however, has not kept up with improvements in network and CPU speed. While both network links and CPUs are purely electronic devices, disks are mechanical. Mechanical considerations such as rotation rate and head positioning speed limit disk performance, as do other factors such as the amount of data that can be stored on a square inch of medium. Rotation rate for most disks only increased from 3600 to 5400 revolutions per minute (RPM) during the period from 1980 - 1994. This is a 50% improvement over 14 years, or just 3% per year. Even the fastest disks currently available rotate at 7200 RPM — an average annual increase of 5%. Smaller disk platters can rotate faster than large ones because they use less power and stress the platter material less. However, disks are unlikely to decrease in size much beyond an inch in diameter — a factor of three smaller than disks common today. Disk seek time is limited by how rapidly an arm can move across a disk's surface and how quickly the head can "settle" on the correct track. As with rotation rate, smaller devices allow faster disk seeks. Nonetheless, disk seek times cannot improve dramatically because disk heads cannot accelerate and move too quickly. While seek time and rota-

tion rate have improved slowly, storage density has grown more quickly. The amount of data that can be stored on a single square inch of a disk platter — the *maximum areal density* (MAD) — has historically followed the curve $MAD = 10^{(year-1971)}$ million bits per square inch [40]. This density, however, is the product of tracks per inch and data stored along a track in a single inch. Thus, data per disk track for a constant diameter disk increases approximately as the square root of MAD. A disk's sustained transfer rate is proportional to the amount of data stored on a single track; while the disk may be capable of much higher peak rates, it is limited in the long term to transferring data that passes under its read/write head. Disk performance is discussed more completely in Section 2.1.

6.1.1. Network Performance

Since RAMA distributes data to disks pseudo-randomly, it relies heavily on the MPP interconnection network to move data from where it is stored to where it is needed. RAMA must not use the network too heavily, however, as applications running on the MPP also need to use the network to pass data between cooperating processors. This section describes the results from experiments varying two network parameters — network bandwidth and network message latency. Variations in these parameters are considered for several applications running on each of two possible network configurations.

One network configuration tested is a mesh network, a typical MPP interconnection scheme. In a mesh, each processor has a link to its nearest neighbors in 4 directions. The mesh tested was a torus, as depicted in Figure 5-4.

In addition, some simulations assumed a star network, in which each node has a single connection to a central hub with very high bandwidth. The hub must provide a full-speed connection at any time between any two nodes. This requires bandwidth equal to (*linkBandwidth* × *numNodes*)/2. A message from one node to another takes exactly two hops: one from source node to hub, and the other from hub to destination node. A workstation network using a star network might have long network message latencies (1 ms) and moderate link bandwidth (10-30 MB/s per client). This section shows that the RAMA design achieves high bandwidth on workstations connected by high-speed high-latency links as well as on MPPs.

6.1.1.1. Network Bandwidth

RAMA relies on high network bandwidth to move data between the disk the data is stored on and the MPP node requesting it. Links of 100 MB/s allow RAMA to run at full speed without congesting the interconnection network. While RAMA does not suffer much of a performance loss at lower bandwidths, it does cause more congestion. Since MPP applications make heavy use of the interconnection network, they run slower when the file system places a heavy load on the network links.

Figure 6-1 shows the total time needed to read a 32 GB file and the average load per network link for various link bandwidths on a 16 x 8 processor mesh. It takes less than 1% more time to read a file with 10 MB/s links than it does with 200 MB/s links. However, average link load is, as expected, 20 times higher for the 15 MB/s links than for the 200 MB/s links. If the network links are, on average, 30% loaded by the file system, any applications that use the interconnection network will likely run slower due to link contention. Thus, low bandwidth links are not acceptable for MPPs running large scientific applications. However, a 30% load on the network links would be acceptable for workstation file service since file data is normally a large component of network traffic in such systems. Additionally, most workstation programs do not rely heavily on high-bandwidth low-latency network communications and would not be adversely affected by the relatively high network load.

Writing a 32 GB file produces a curve similar to that of the 32 GB read, as Figure 6-2 shows. Again, file system bandwidth remains relatively constant while link bandwidth drops from 200 MB/s to 10 MB/s. The disks, not the network links, were the bottleneck in both the read and write cases. While a 10 MB/s network

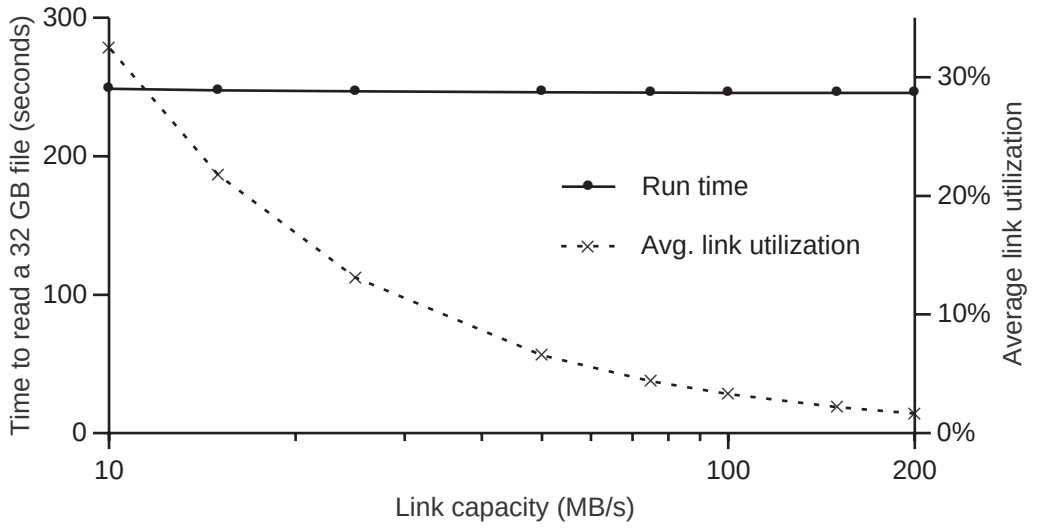

Figure 6-1. Effects of varying link bandwidth in a mesh interconnection network on RAMA read performance.

This figure graphs the time needed to read a 32 GB file from a RAMA system on a 16 x 8 processor mesh with a solid line. Note that total elapsed time hardly varies with link bandwidth. Likewise, the average amount of data transmitted across a link does not decrease markedly as maximum link bandwidth increases. However, average link utilization, shown by the broken line, drops off dramatically as maximum link bandwidth increases. Each link carries approximately the same absolute load regardless of its maximum bandwidth. However, the interconnection network is less congested by the file system if there is more bandwidth available.

takes 100 ms to deliver a megabyte of data, a single disk takes at least half a second to read or write the same data. A single one megabyte file request is split into many smaller requests, each sent to a pseudo-randomly chosen disk. Since each node in the MPP is doing the same thing, each disk receives many small I/O requests. As network speed decreases, some of the disk I/O requests are delayed. However, those same I/Os must wait even longer for the disk to finish servicing the first few requests. Thus, the file system's speed is limited by disk bandwidth. As long as the interconnection network remains close to an order of magnitude faster than disk, RAMA will perform at full efficiency for large reads and writes.

The results of a simulation using a workstation-style network are shown in Figure 6-3. This set of simulations uses a star network (also called a hub-and-spoke network) rather than a mesh network to connect all of the nodes, more closely resembling a network of workstations. The RAMA design provides the same bandwidth in this configuration as in the mesh configuration, but average link utilizations are lower for a star network. In an $x \times y$ mesh network, each message between client and disk must traverse an average of $x/4 + y/4$ links. If the overall file system delivers b MB/s of data, each link must carry $b\,(x/4 + y/4)\,/\,(2n)$ MB/s, where the MPP is composed of $n = xy$ nodes. In a star network with n nodes, however, each message only crosses two links — the link to the client node and the link to the disk node. Since there is only one connection to each node, however, each link carries $2b/n$ MB/s. A star interconnection network will thus run RAMA better than a mesh network with the same number of nodes and the same link bandwidth whenever $(x + y) > 16$. However, mesh networks scale much better than star networks. In a star network, all traffic must pass through a central hub. Building a hub that can support sufficiently high bandwidths between any two of hundreds of nodes is expensive, as the hub may require additional components to support the added bandwidth. On the other hand, adding nodes to a mesh simply

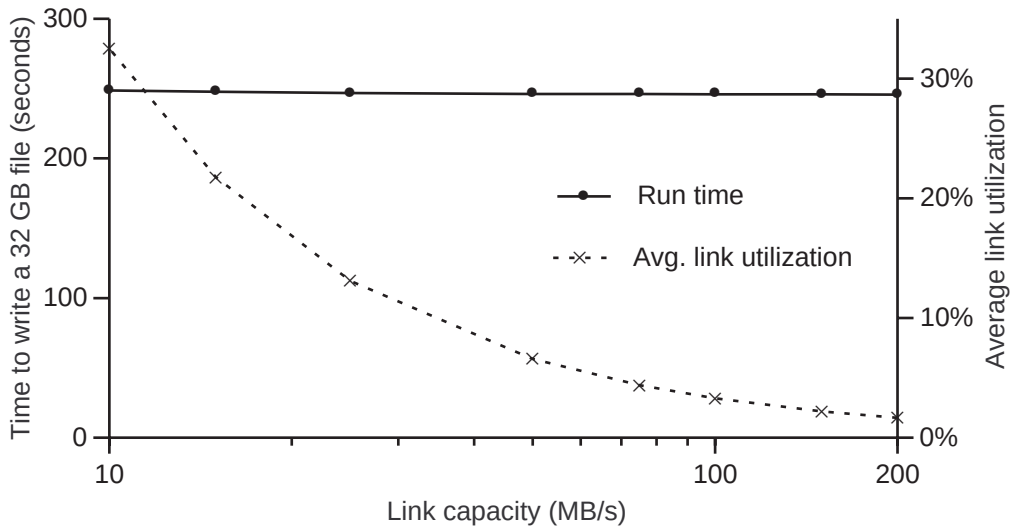

Figure 6-2. Effects of varying link bandwidth in a mesh interconnection network on RAMA write performance.

The RAMA design's write performance is, surprisingly, similar to its read performance, even for low network bandwidths. While slow network links delay the arrival of most of the data written at the destination node and disk, the disk is still the bottleneck. Even with a 10 MB/s network, most of the written data arrives before it would have been written with the fast network. The disks thus see little difference between a network with 10 MB/s links and one with 200 MB/s links.

involves replicating a basic network interface. Nonetheless, the RAMA design performs sufficiently well on a relatively slow star network to permit its use as a file system for a network of workstations.

6.1.1.2. Network Message Latency

Besides link bandwidth, another important network measure is the overhead required to send a packet of information from one node to another. Most of this overhead is due to software, particularly managing the protocol stack and routing messages. Modern computer systems can take anywhere from 1 to 2000 microseconds to send out a single message. This delay is largely a software delay caused by a multi-layered network protocol stack. Fortunately, this penalty is usually paid only once per packet, even if the data is transmitted as many physical-layer packets and must traverse several network links.

The network model used in the RAMA simulations charges the overhead at the source node before the data is placed into the network. Since this overhead is typically a software delay, the physical network link is not considered in use during this period. Incoming or traversing message may thus use the link while an outgoing message is in the process of being sent. Long message latencies can potentially lower overall file system performance by increasing the total response time for I/O requests. In particular, a read incurs two delays — one in sending the request, and the other in replying with the requested data.

The RAMA simulations, however, show that message latency has little effect on file system performance on large files, as file system performance is disk-limited. Figure 6-4 shows RAMA's performance as network message latency increases from 5 μs to 2500 μs. The increase in message latency causes less than a 2% increase in the time needed to read a 32 GB file. Since the total elapsed time and total bandwidth show little change, link utilization also varies little as message latency increases.

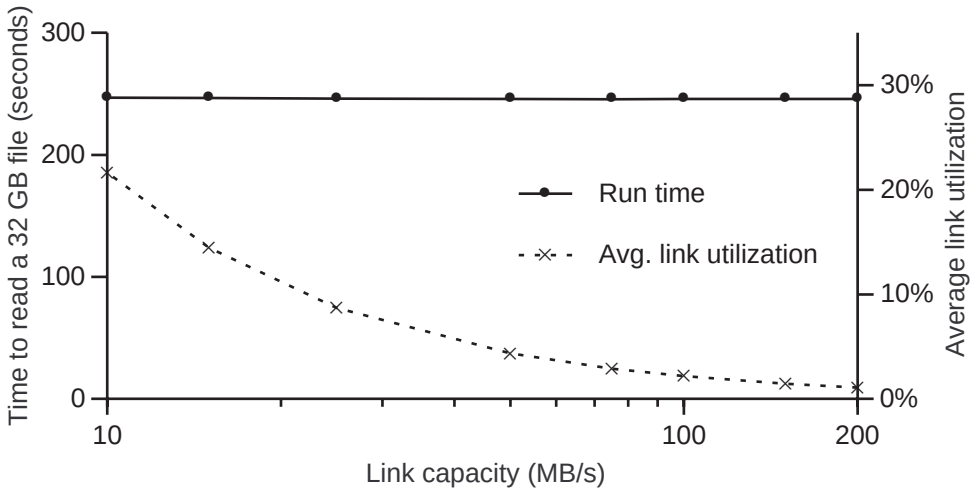

Figure 6-3. Effects of varying link bandwidth in a star interconnection network on RAMA performance.

This graph is similar to that in Figure 6-1, but substitutes a 128 node 100 MB/s star network for the mesh network in the previous figure. Again, file system performance varies little even as link bandwidth increases from 10 MB/s to 200 MB/s. Link utilization is lower than in the mesh case, however. While the mesh has twice as many network links as the star, each message in the mesh must, on average, traverse six links as compared to two in the star network.

This lack of variation is unexpected, as two delays of 2.5 ms each should add 5 ms to the length of each file request. However, there are two reasons why the variation due to message latency is so small. First, average response time for an individual disk request — in this case 32 KB — is nearly 300 ms. While requests use the disk for only 18.45 ms each, every processor issues 32 requests to various disks to read a total of 1 MB from disk every iteration. For a 128 processor system, this results in 4096 requests being issued at once, for an average of 32 requests per disk. Since the all requests are made at the same time, each request will wait, on average, for half of the 32 requests for its disk to finish before it begins service. This incurs a delay of approximately 280 ms which, when added to the average service time of 18 ms, yields a response time of just under 300 ms. An increase of 5 ms in a 300 ms request is less than 2%, in contrast to the 27% increase from a 5 ms increase to an 18 ms request.

Additionally, the disks themselves distribute replies over time. This effect is similar to the one that occurs for slower network links. The entire file system request is delayed by the message latency from the last disk request to complete. However, the latencies for the other messages do not increase the time needed to complete the file system request. While these latencies do make each individual message take longer, only the finishing time of the last disk request determines when the entire file system request finishes. The data from the other disk requests will be available to the requesting node, regardless of message latency, before the arrival of the data from the last disk request to complete.

6.1.2. Disk Performance

Disk performance is another technological factor that drastically affects file system bandwidth in RAMA. The three main determinants of disk performance are rotation speed, data density along a disk track, and seek time. Certainly, disk transfer bandwidth is also an important factor in disk performance. However, sustained disk bandwidth is limited to the amount of data that passes under one or more read-write heads each second. For the majority of disks today that use only a single active head, the maximum sustained bandwidth is the product of the number of bytes stored on each track and the number of rotations per second. This

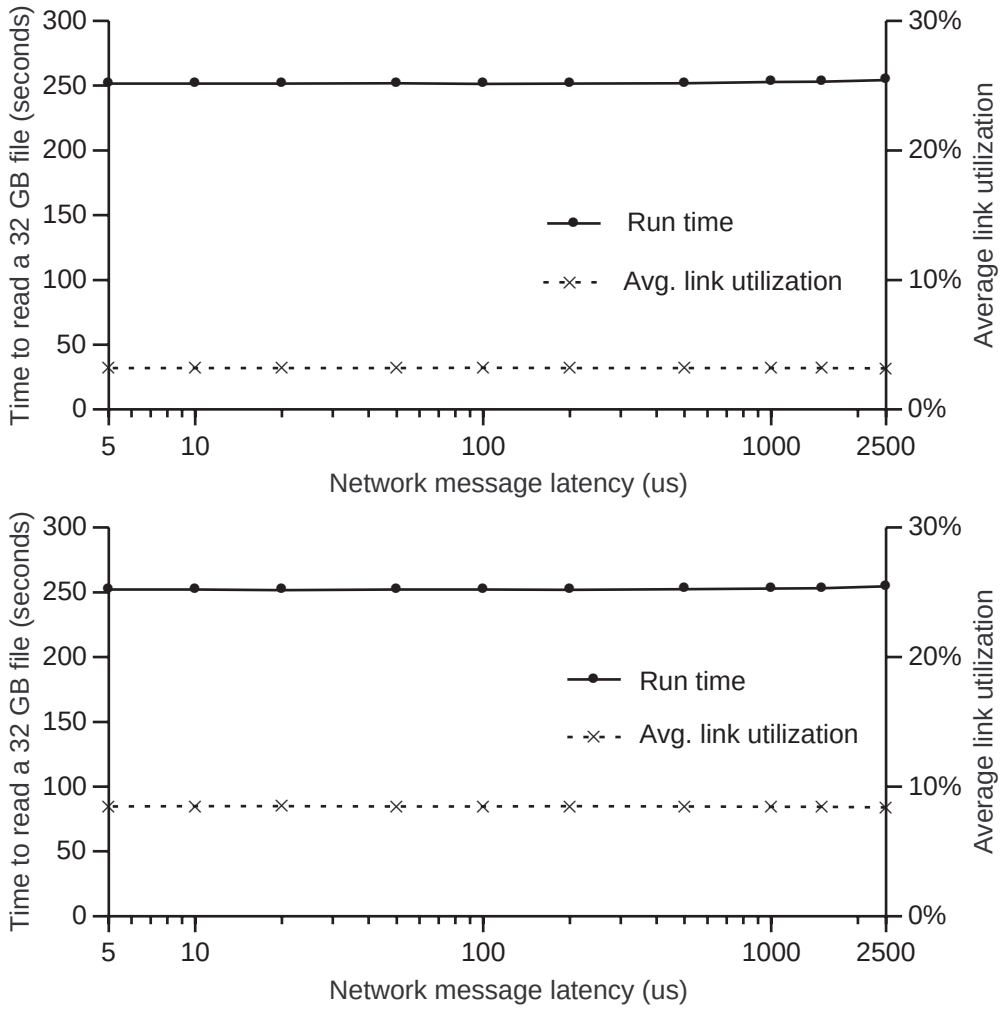

Figure 6-4. Effects of varying network message latency on RAMA performance.

As network message latency increases, individual messages take considerably longer to be delivered. However, the limiting factor is still disk bandwidth. The top graph shows RAMA's performance on a 16 x 8 mesh network with 100 MB/s links. As message latency on a increases from 5 µs to 2.5 ms, overall bandwidth and average link utilization increase less than 2%. The lower graph shows similar results for a 128 node star network with 25 MB/s links.

simple formula does not include track-to-track seek time and head switch time. These operations require 1-2 ms on modern disks, reducing maximum sustained bandwidth further by 5% - 20%, depending on rotation rate. This section thus does not use sustained bandwidth as a basic parameter; instead, it is a measure derived primarily from rotation speed and disk track density.

Figure 6-5 shows the contributions of each performance characteristic to the total time required for a disk operations of various sizes. The three disk parameters are orthogonal to each other — a disk manufacturer may improve any or all of the performance metrics while leaving the others unaltered. Improving only a single metric eventually results in little gain, as the contributions of the other two parameters dominate overall performance. The following sections describe each parameter in detail, and discuss their effects on RAMA's simulated performance.

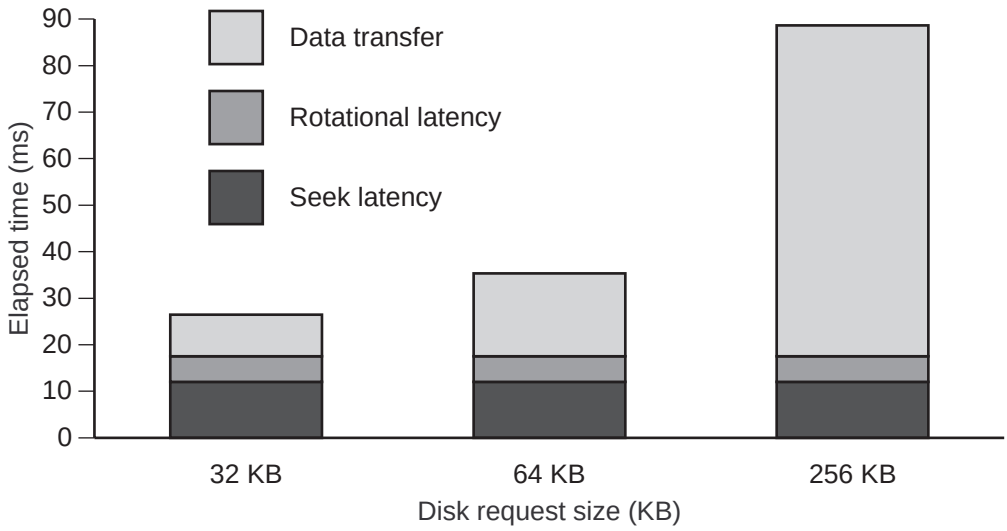

Figure 6-5. Components of disk latency in a single RAMA disk request.

Each of these columns represents the total time needed for a single disk I/O in RAMA. The columns are broken up into the components of the request service time, showing the delays due to seek latency, rotational latency, and data transfer. For larger requests, transfer time dominates. However, positional latency (seek and rotation) become more important as the amount of data transferred decreases.

This thesis does not consider disk buffering because it would not speed up file service for supercomputer workloads. Traditional Unix file systems gain great advantage from disk track buffers because the file system issues many separate but contiguous small reads and writes. Thus, buffering a full track in the disk's cache allows the disk to transfer sequential requests after the first at peripheral bus speeds which are at least two or three times faster than sustained transfer rate. However, RAMA does not request contiguous blocks in separate disk I/O requests. Rather, it issues a single large request for all of the data to be read or written to disk. This makes a disk cache unnecessary. While disk buffering may improve RAMA's performance on a small file, small request workload, it will have little effect on large file workloads.

6.1.2.1. Disk Track Density

Increasing the areal density of data on disk can result in both more data per track and more tracks per inch as data is packed more densely along both the diameter and circumference of the disk. However, sustained disk bandwidth depends primarily on how much data is read in a single revolution of the disk. Linear density increases as the square root of areal density because areal density also increases the number of tracks per inch. Since sustained bandwidth is proportional to linear density, recent increases in disk capacity have not produced similarly large increases in disk bandwidth.

Because track density affects sustained bandwidth, however, it has an impact on RAMA's performance. Figure 6-6 shows the effects of increased track density, in the form of higher bandwidth, on RAMA's expected throughput when reading a large file. Initially, the faster transfer rate improves performance. However, this increased bandwidth reaches a point of diminishing returns when the other factors in disk access time — head latency and rotational latency — dominate the total access time. The point at which the falloff occurs can be delayed in two ways.

First, the amount of consecutive data per disk in RAMA can be increased. A request for the data from a single disk consists of head positioning followed by data transfer. Maximizing the amount of data trans-

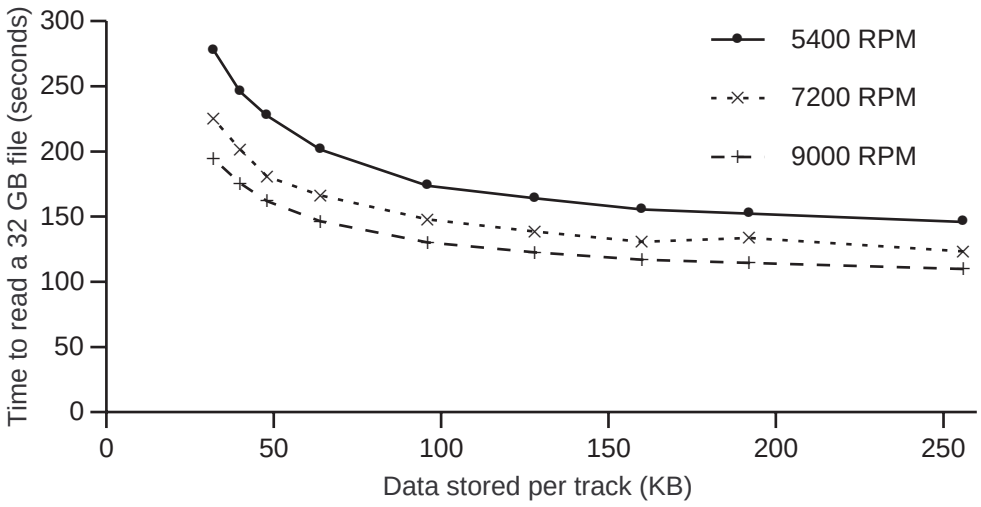

Figure 6-6. Effects of varying disk track capacity on RAMA performance.

Sustained transfer bandwidth is directly proportional to the amount of data on a single track. As the data on a single track increases, bandwidth likewise increases. However, file system bandwidth soon becomes limited by seek and rotational latency, as transfer time for the 32 KB units in these simulations grows shorter. Past 150 KB per track, increased density provides little performance advantage without corresponding increases in other parameters.

ferred for each head position increases the fraction of the time the disk spends transferring data and thus increases total bandwidth. However, increasing the amount of data stored on each disk decreases the number of disks involved in a single file system request, reducing concurrency. Section 6.2.1 explores these tradeoffs further.

Improving seek time and rotation rate will also allow higher track densities to improve performance, as transfer time initially consumes a larger portion of the access time. Unlike the first solution, however, this approach requires improvements in hardware technology.

6.1.2.2. Disk Rotational Speed

Disk rotation speed has two effects on RAMA's performance. First, it improves disk bandwidth, thus reducing transfer time. In addition, faster rotation speeds decrease rotational latency, allowing the disk's read/write head to reach the necessary data faster. Thus, higher rotation speeds decrease two of the three components of disk access time shown in Figure 6-5.

Increasing disk rotation rate improves RAMA's performance, as Figure 6-7 shows. As with other performance metrics, however, steadily increasing the rotation rate reaches a point of diminishing returns. The curve is slightly steeper than that for disk track density, however. Increasing track density only decreases transfer time, so an infinitely high density would be limited by both seek time and rotational delay. On the other hand, a high rotational speed improves both bandwidth and rotational delay, leaving overall throughput limited only by head seek delays. Since rotation rate affects two components of access time, the performance falloff is slightly steeper and continues longer than for track density.

Unfortunately, increasing disk rotation speed is perhaps the most difficult challenge for disk designers. Disk rotation rates have only increased from 3600 RPM in the mid 1970's to 7200 RPM today — a factor of two improvement in twenty years. In the past, faster rotation speeds posed both signal processing problems and

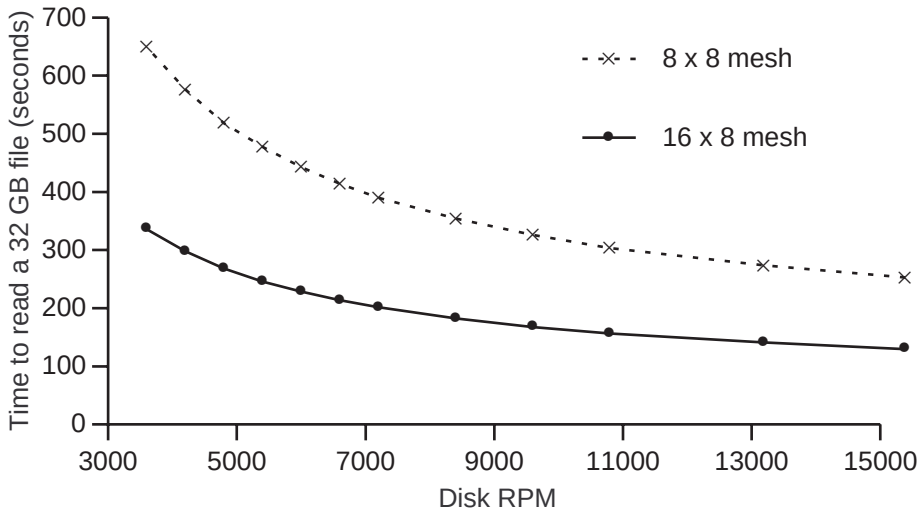

Figure 6-7. Effects of varying disk rotational speed on RAMA performance.

This graph shows the effect of increased rotation rate on RAMA's simulated performance. All disk parameters except for rotation rate are taken from the standard disk model used in the simulations. Each curve shows the results of simulations varying disk rotation rate from 3600 RPM to 15,200 RPM. A four-fold increase in rotation rate results in more than double the bandwidth for an 8 x 8 mesh, but just under double the bandwidth for a 16 x 8 mesh. The difference is due to the longer queue lengths, and resulting shorter average seeks, from fewer disks.

basic physical difficulties. Today's drive electronics can cope with the signal processing demands of the higher bit rates from rapidly rotating drives. However, basic mechanics limit safe rotational speeds for metal disks, particularly as read/write heads float ever-closer to achieve higher data density. It is unlikely that disks will exceed 20,000 RPM in the near future, but even that speed is only three or four times current rotation rates. Thus, slightly higher disk rotation rates are unlikely to provide great improvement in RAMA performance in the next few years.

6.1.2.3. Disk Head Positioning Latency

Another important factor in disk performance is the time required to move the disk's read/write head from one track to another. Like rotation rate, head positioning time is primarily a mechanical operation, and thus has only limited prospects for great improvements over time. While seek times have improved more than rotation rates over the past twenty years, these gains have come from smaller read/write heads and smaller disks.

Current disks have an average seek time under 10 ms and a minimum track-to-track seek time of 1 - 2 ms. While the average seek time is still dropping, minimum seek time is limited by the need to accelerate and decelerate the head and settle on the proper track. Maximum seek times, too, are decreasing as disks and heads shrink. However, the maximum seek time is limited by the need to move the head several centimeters. Already, the head must accelerate to over 2 meters per second, move 3 or more centimeters accurate to within 0.03%, and decelerate. While average and maximum seek times will continue to drop, the match the improvement in disk data density.

At first glance, it might seem that increasing the number of cylinders without increasing disk size would improve disk performance. However, this is not the case for RAMA. Both average and maximum seek time are largely unaffected by the number of tracks the head must traverse because the largest components of

seek time — acceleration and constant velocity motion — are independent of track density. Deceleration. and settling time do increase slightly as track density increases, but their effect on average and maximum seek times is not large. Though an average seek can reach more data with more cylinders per inch, individual files do not benefit greatly. A single file must occupy several consecutive cylinders to benefit from more densely packed cylinders. However, each cylinder holds large quantities of data — almost 500 KB for the base disk used in the RAMA simulations. As Section 6.2.1 shows, storing large sections of a file contiguously on a single disk leads to worse performance because of the loss of concurrency.

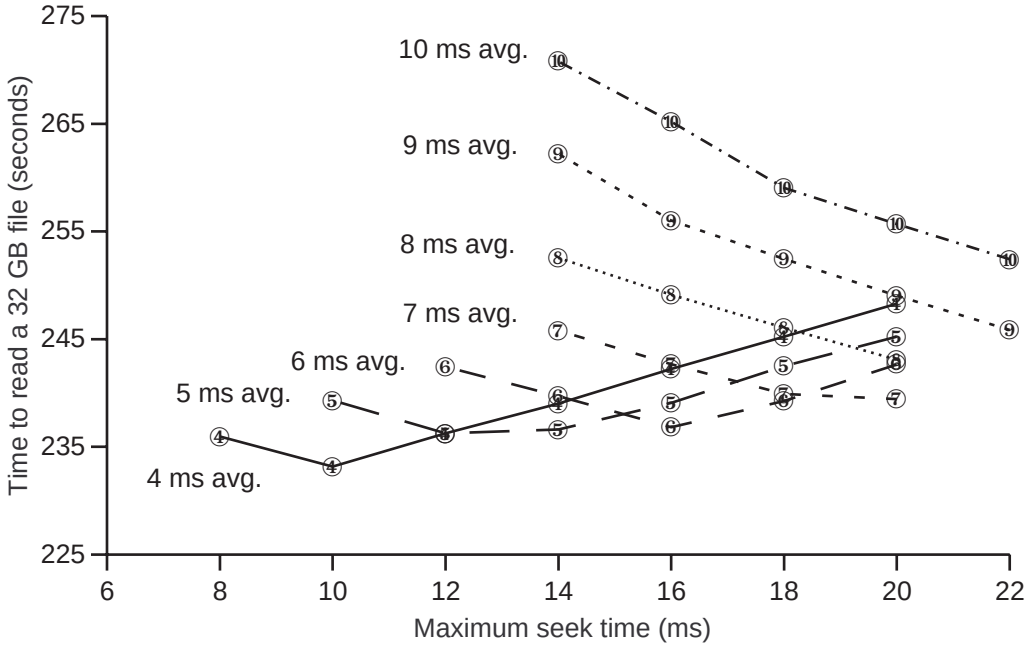

Figure 6-8. The effects of average and maximum disk head seek time on RAMA performance.

This figure shows the effects of different average and maximum seek times on projected RAMA performance. Each line represents simulations with the same average seek time, varying maximum seek time along the horizontal axis. Simulations covered average seek times from 10 ms down to 4 ms, and maximum seek times between 8 and 22 ms.

In general, lower average seek times resulted in better performance. Surprisingly, however, file system performance worsened in many cases where maximum seek time decreased. This counterintuitive result is explained in Section 6.1.2.3.

As expected, lower average seek times resulted in better RAMA performance in simulations. Figure 6-8 shows this effect, as average seek times from 10 ms down to 4 ms were simulated. Each line in the graph represents a series of simulations with the same average seek time and different maximum seek times. As the graph shows, however, varying seek times produce relatively limited gains — approximately 25% between the best case at 4 ms average seek time and the worst case at 10 ms average seek time.

The increase in performance from faster average seek times is hardly unusual. However, Figure 6-8 also shows that RAMA performance *decreases* in many simulations with lower maximum seek times. For example, holding average seek time constant at 10 ms while increasing maximum seek time from 14 ms to 22 ms results in a 6.5% performance increase. This counterintuitive result stems from the two factors in the time needed to seek from one track to another — acceleration time and constant velocity seeking. Figure 6-9

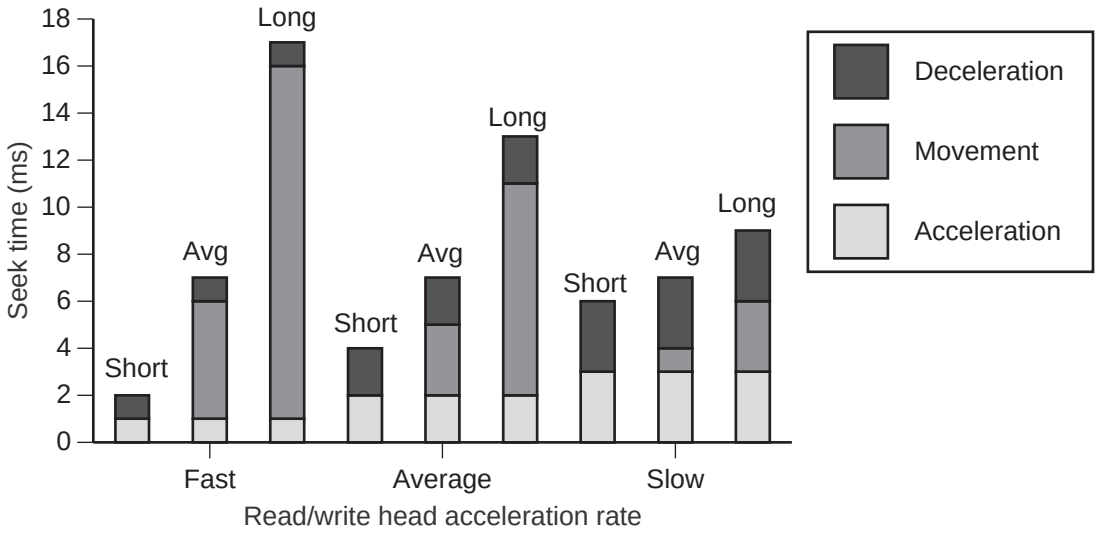

Figure 6-9. Acceleration and travel components of disk read/write head seek time.

This diagram shows the contributions of acceleration, constant velocity motion, and deceleration to total disk head seek time. Short, average, and long seeks are shown for several choices of maximum seek time with the same average seek time. This highlights the performance differences between the cases. Disks with maximum seek time close to average seek time perform relatively well on long seeks, but poorly on short seeks. Disks with higher maximum seek times do better on seeks considerably shorter than average, but worse on long seeks.

shows the acceleration, movement, and deceleration components for disks with different disk arm acceleration rates. If maximum seek time is just less than average seek time, as in the 10 ms average, 14 ms maximum case, the disk head must accelerate relatively slowly but traverse the surface rapidly. On the other hand, a larger difference between average and maximum seek implies that acceleration and deceleration are relatively rapid, but the disk head moves more slowly at top speed across the surface. Current disks are closer to the first model than the second, since most disks have average seek times 1/2 to 1/3 of their maximum seek times. RAMA, like most file systems, attempts to make disk seeks cover as few tracks as possible. Thus, acceleration rate is more important than high speed on longer seeks. Disks whose average seek times are considerably smaller than its maximum seek time, including most modern disks, will perform well in such an environment.

6.1.3. CPU Speed and Memory Size

CPU speed and the amount of memory on each node of an MPP do not themselves have a major effect on RAMA's performance. Transferring 32 KB of data in RAMA uses fewer than 10,000 instructions. Since RAMA's peak bandwidth per node is under 2 MB per second with current disk technology, each node requires fewer than 6 million instructions per second to service all of the file system requests. Additionally, processors are increasing in speed faster than disks are, so the percentage of CPU cycles used for the file system will drop in the future.

Additional memory on each node will help the file system somewhat, as RAMA can cache more metadata to improve disk performance. Caching file data is of little help, as I/O-intensive MPP applications do not exhibit much file locality. Additionally, the files are so large that reasonable amounts of memory would not be able to hold sufficient file data to provide much improvement. For example, a $65{,}536 \times 65{,}536$ double precision matrix requires 32 GB of storage space. Even on a 256 node MPP, this is 128 MB of data per node.

However, Chapter 7 will show that large matrix decompositions perform well with as little as 8 MB of storage space per node. Additional file caching provides no performance improvement for this algorithm since the entire matrix must be read again before an individual file block is rereferenced. Unless the file cache can hold the entire matrix, it will not speed up the file system.

Caching metadata, on the other hand, may result in better performance. The RAMA simulator used in this thesis did not model metadata accesses. However, fully caching line descriptors for a 1 GB disk requires approximately 4 MB of memory. If an MPP node can afford the space for this cache, metadata access will not reduce file system performance. Extra memory on a node also allows RAMA to cache directories, thus reducing the time necessary for directory searches, as discussed in Section 6.3.

While additional memory and faster CPUs do not much affect file system speed, they have a great effect on applications that may use the file system. Faster processors allow scientists to use more detailed models and simulate them in less time. Larger memories do not usually speed up applications, though they do permit the solution of larger models by keeping programs CPU-bound rather than I/O-bound.

Matrix decomposition, as described in Section 5.2.2, is an application for which faster processors may be exploited for either faster solution time or to the solution of larger problems. Figure 6-10 shows the amount of time needed to decompose variously-sized matrices on an 8×8 processor mesh composed of nodes with varying memory sizes and CPU speeds. For all CPU speeds and matrix sizes, sufficiently large per-node memories keep the algorithm CPU-bound. However, larger matrices require more memory to keep the application CPU bound, as do faster CPUs. Adding additional memory results in no performance improvement once the algorithm is CPU-bound. Traditional MPP designers add as much memory as possible to their systems to allow programmers to avoid using I/O. One reason for this is the difficulty in achieving good file system performance — many file systems require the programmer to specify the layout on disk to use the file system efficiently. By addressing this problem, RAMA allows MPPs to use file system bandwidth to replace larger memories.

Global climate modeling is another application whose I/O demands increase with CPU speed. Since GCMs perform hundreds or thousands of floating-point operations for each grid cell, they require less memory. For example, a GCM may track 100 variables in each cell at 20 altitudes. This uses only 16 KB of memory per grid cell, yet a GCM may perform over 500,000 floating-point operations to model twelve hours of its behavior. As a result, current MPPs are not fast enough to run long-term models larger than 200×200 grid cells. Even such a large model would only require 640 MB of memory, or less than 8 MB per node in a 128 node machine. The GCM studied in this thesis uses the file system to store data from each iteration; the bandwidth required is directly proportional to the speed of each CPU. Since the application is already using nearly the maximum bandwidth that RAMA can provide, increasing the CPU speed will only result in an I/O-bound application. This difficulty can be alleviated either by using more disks per processor or by getting faster disks. Since increases in disk transfer rate will not keep up with the gains in CPU speed, however, providing I/O for GCMs that write detailed logs will be difficult.

6.2. Design Parameters

One major advantage that RAMA enjoys over conventional parallel file systems is that it has far fewer configuration options. Data is distributed pseudo-randomly, so the only parameters for distributing data to disks are a hash function and the amount of consecutive file data placed on each disk. While any hash function that generates a pseudo-random number from a block offset and file identifier will suffice, varying the amount of file data stored on each disk affects RAMA's performance. Network and disk performance, too, are fixed by the technology used to build the system. However, the type of network used and the number of disks per CPU are both parameters that can be changed.

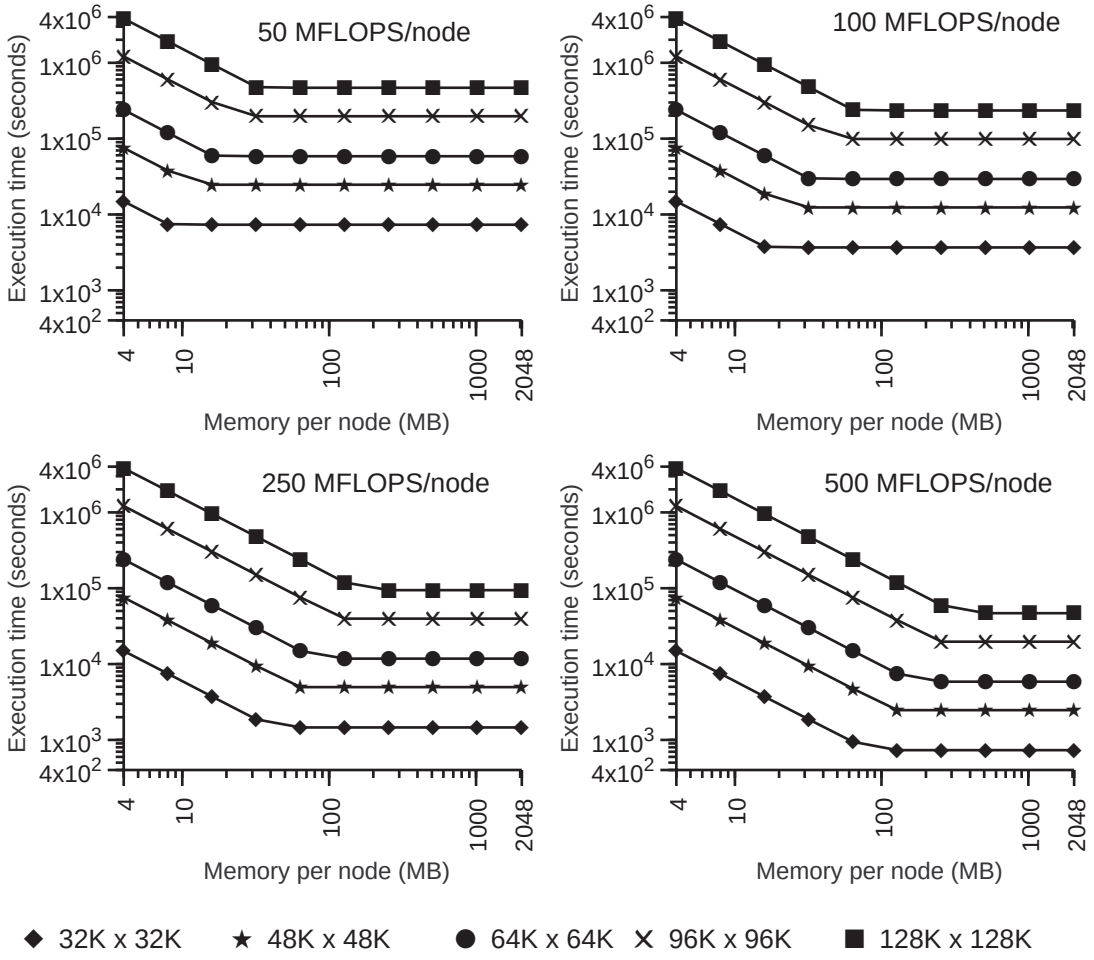

Figure 6-10. Matrix decomposition performance with varying memory sizes.

This figure shows the performance of LU matrix decomposition on an 8×8 processor mesh with varying amounts of memory. Each graph shows the run times for various matrix sizes and per-node memory sizes at a constant MFLOPS rate. The graphs show execution times for multiprocessors with 50 MFLOPS, 100 MFLOPS, 250 MFLOPS, and 500 MFLOPS per node.

The LU decomposition algorithm is able to overlap almost all I/O with computation, so program execution time is constant if the algorithm requires less bandwidth than the file system can supply. When run with small memories, however, the algorithm becomes I/O-bound and performance suffers.

6.2.1. Data Distribution on Disk

A conventional striped file system requires much configuration information to provide optimal performance. Determining the best pattern in which to lay the data out on disk is a difficult task — the file system designer must decide how many processor nodes have disks attached to them, how many disks each node has, and the pattern in which data is distributed among all of the disks. RAMA, on the other hand, relies on randomness to provide good performance. The only data distribution parameter that affects RAMA's performance is the choice of how much data from a single file to store on a disk before moving to the next randomly-chosen disk.

The hash algorithm in RAMA determines data placement on a per-file block basis. Each file block may be located on a randomly-determined disk, if desired. However, such behavior provides poor performance because the seek and rotational latency overheads occur for each file block. Thus, RAMA's hash algorithm can group consecutive file blocks on the same disk, allowing the positional latency to be amortized over more data.

Figure 6-11 shows RAMA's projected performance, using current disk technology, for varying amounts of data stored consecutively. As expected, simulations that store few consecutive file blocks on a disk exhibit poor performance. However, performance is also worse for very large amounts of data stored consecutively. If RAMA stores too much file system data on each disk, concurrency is much lower. For example, a program running on a system with 128 disks might have 32 MB of I/O requests outstanding at any time. This corresponds to requests to 1024 requests of 32 KB each, but is only 64 requests of 512 KB each. In the former case, each disk will average 8 requests; though some will have more and some fewer, every disk will be nearly fully utilized. In the latter case, however, at least half of the disks will be unused at any time, limiting file system bandwidth to half maximum. Worse, some disks in this system might randomly receive two or even three requests, further hurting performance.

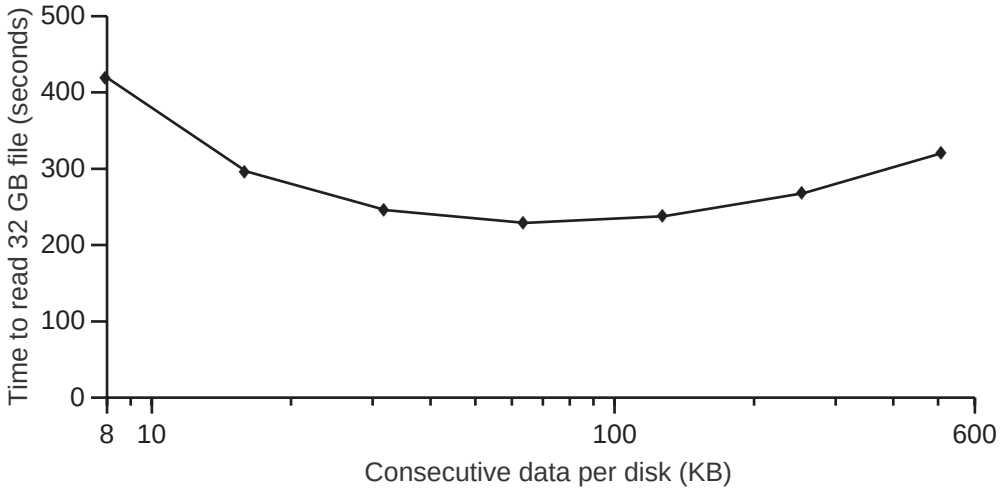

Figure 6-11. Effects of varying the amount of consecutive file data stored per disk.

This figure shows the effect on performance of varying the amount of consecutive file data stored on each disk in RAMA. The file system's hash algorithm assigns several consecutive file blocks to the same disk line, where they can be read and written sequentially. If too few blocks are grouped together, access latency dominates and performance suffers. However, grouping too many blocks together also adversely affects performance. If too much consecutive data is stored on the same disk, concurrency is reduced and fewer disks are active at a given time. While individual disks may perform better in this case, the file system as a whole suffers by leaving some disks idle.

The curve shown in this graph is applicable only today's disks. Section 6.4 and Figure 6-14 discuss similar curves for possible future disks.

This issue is not unique to RAMA. Striped file systems face the same problem, particularly as individual stripe units become very small or very large. A striped file system with very large stripe units suffers the same loss of concurrency that RAMA does. Additionally, a striped file system using small stripe units pays relatively larger head positioning and rotational latency penalties similar to those incurred in RAMA. While

RAMA can pick an optimally sized “stripe unit” without compromising performance for various access patterns, however, striped file systems cannot. This issue is discussed further in Chapter 7.

6.2.2. Scalability and Multiple Disks Per Node

Another issue that the RAMA design brings up is how many disks to attach to each processing node. Most of the simulations in this thesis assume a single disk per node, but that is certainly not the only choice. RAMA can provide higher disk bandwidth using several disks attached to each node. Since data is distributed pseudo-randomly, clustering multiple disks on every CPU node does not introduce load imbalances.

A closely related problem is that of scalability. Any two MPPs running RAMA with the same number of disks should, in the absence of CPU or network bottlenecks, perform similarly. For example, a system with 32 nodes and 4 disks per node should provide the same bandwidth that a system with 128 nodes and 1 disk per node provides. Additionally, a system with twice as many disks should provide twice the bandwidth. If a file system cannot approach these numbers, it is not truly scalable and may not be suitable for use in scalable MPPs.

As Figure 6-12 shows, RAMA’s behavior with a single disk per node scales quite well. For a 4 x 4 mesh, total bandwidth is 18.5 MB/s while reading a 36 GB file. At 8 times the size, on a 16 x 8 mesh, bandwidth for the same transfer is 148 MB/s. This is almost exactly eight times the bandwidth of the smaller system, scaling up well from 16 nodes to 128 nodes. Note, however, that bandwidth falls off as the number of disks per node increases beyond one. This occurs for two reasons. First, the variation in the maximum number of requests any disk will have to satisfy during a single iteration of the algorithm increases as the number of disks increases. If the algorithm attempts to read 32 MB each iteration, 1024 individual 32 KB disk requests will be outstanding. If those 1024 requests are distributed among 32 disks, the average disk will service 32 requests; however, one disk may serve as many as 40. If the number of disks is doubled, the average requests per disk is cut in half to 16, but the expected maximum may be 22. Since all of the nodes must wait for the slowest node, the overall bandwidth does not increase by a full factor of two. This factor becomes particularly important as more disks are added to each node. When each node has eight disks, every disk averages only four 32 KB requests, yet some nodes may serve more than twice that number. This lack of concurrency results in poor speedups for multiple disks.

Serving multiple applications simultaneously or otherwise increasing the number of outstanding requests will alleviate the problem. The bottom graph in Figure 6-12 shows the same full speed transfer as the top graph, but with each node making 4 MB requests instead of 1 MB requests. Here, adding several disks per node does not cause as steep a drop as in the first case, confirming the reason behind the falloff.

The two graphs in Figure 6-12 also show that two RAMA systems with the same number of disks provide similar performance. The performance levels are not identical because the workloads for each MPP size are not entirely identical. In both graphs, the total amount of outstanding I/O is proportional to the number of nodes in the MPP, as each node reads a fixed amount of data each iteration. For the reasons already mentioned, this difference causes the system with less concurrency — the smaller system — to perform slightly worse. Again, the problem is more pronounced in the top graph, as the concurrency levels are lower there than in the bottom graph. This result supports our hypothesis that RAMA will perform well in multitasking environments with high concurrency.

6.2.3. Network Configuration

While network configuration is an important consideration for MPP designers, it is less so for RAMA. As Section 6.1.1 shows, RAMA’s primary demand on a network is bandwidth. Since the file system pseudo-randomly distributes data, the best configuration is one that keeps the average distance between two nodes as low as possible.

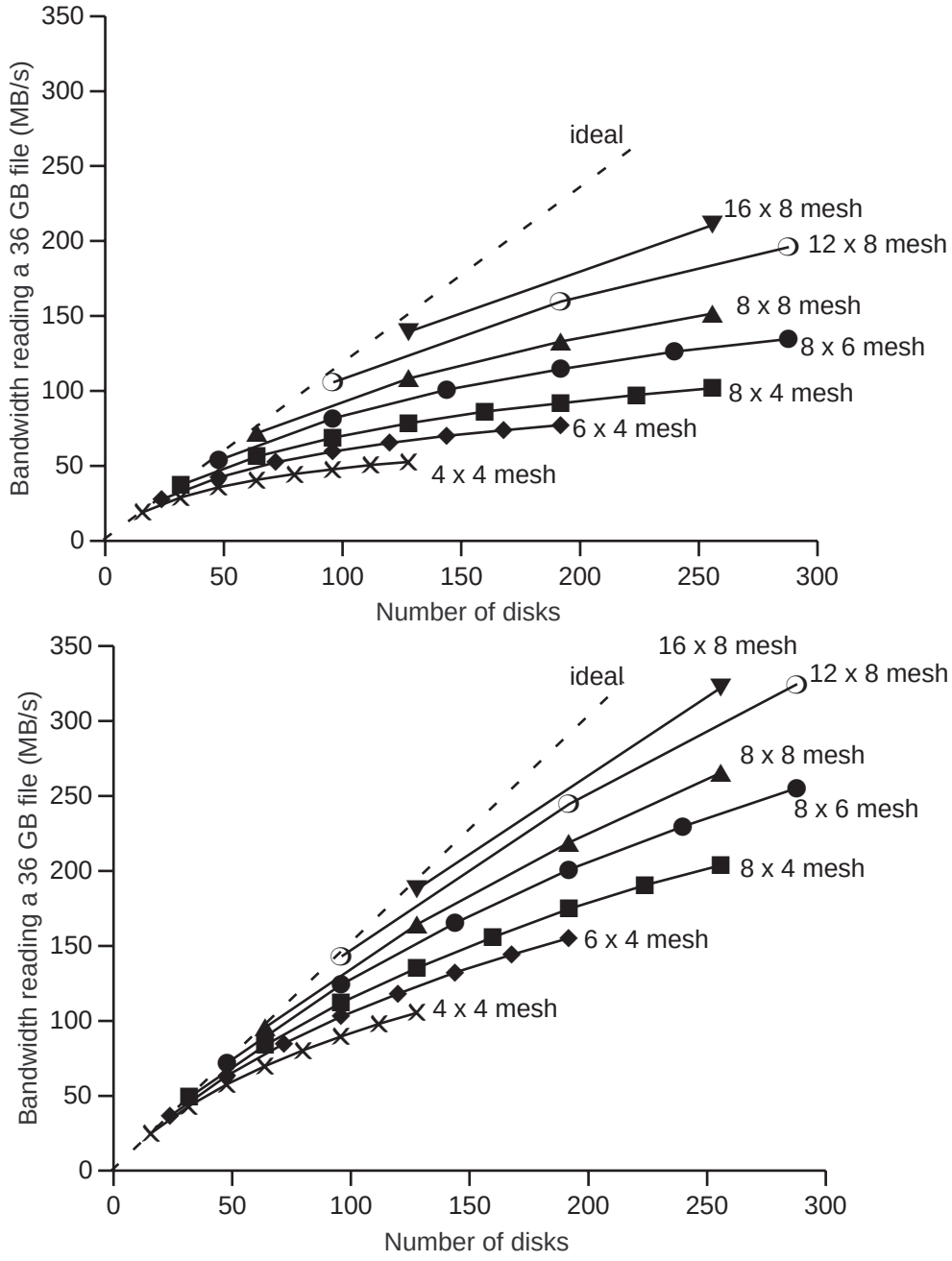

Figure 6-12. RAMA performance with varying numbers of disks per MPP node.

Both graphs show RAMA's simulated performance with varying numbers of disks per node. The X axis in both graphs indicates the *total* number of disks in the system. The Y axis gives the performance in megabytes per second for an application in which all the nodes cooperate to read a 36 GB file. In the top graph, the nodes issue 1 MB reads and wait for all of the nodes to finish the previous I/O before issuing the next one. The situation is identical for the bottom graph, except that the nodes issue 4 MB I/Os instead of 1 MB I/Os.

The expected load that RAMA places on any network is the product of the total file system bandwidth and the average number of hops between two randomly-selected nodes. RAMA distributes this load well in sys-

tems with symmetrical networks, causing the loads shown in Table 6-1. As the table shows, RAMA places reasonable loads even on large networks. For example, RAMA running on a 32 x 32 mesh, with 1024 nodes, will average 16 times the bandwidth of a disk on each node. Since the simulations show that RAMA achieves about 1.2 MB/s per disk for the disks in the simulation, it would put less than 20 MB/s on each link in such an MPP. Moreover, this load can be reduced if the MPP and RAMA file system is subdivided. If the hardware designers decide to use a 3-D mesh instead, RAMA running on a 16 x 8 x 8 mesh will average only 8 times the bandwidth of a single disk — under 10 MB/s. If each link in the interconnection network supports 100 MB/s or more, 10 MB/s per link should be sufficiently low to avoid conflicts with applications passing messages between nodes.

Network type	Average number of hops	Average load per link (MB/s)
Star (n)	2	$\frac{2B}{n}$
Mesh (a x b)	$\frac{(a+b)}{4}$	$\frac{(a+b)}{4ab}B$
3-D Mesh (a x b x c)	$\frac{(a+b+c)}{4}$	$\frac{(a+b+c)}{4abc}B$
h-D Hypercube	$\frac{h}{2}$	$\frac{hB}{2^{h+1}}$

Table 6-1. RAMA's expected load on various network configurations.

This table details the effect that RAMA has on various network configurations while providing *B* MB/s. For all of these configurations, however, the most important factor is the available bandwidth on each link. For MPPs with hundreds of nodes or fewer, the per-link load differs by less than a factor of 10 between different configurations.

6.3. Small File Performance

The previous sections in this chapter have examined RAMA's sensitivity to changes in technological and design parameters using workloads involving large files. However, RAMA must also perform well for small files to permit its use in a system that integrates MPPs and workstations.

The simulation results graphed in Figure 6-13 demonstrate the RAMA design's ability to handle many small file requests using the workload described in Section 5.2.4. This workload consists of many small file requests, each of which reads or writes an entire file whose size is a parameter to the workload. The average request rate for these simulations varies from 5 to 150 requests per second per MPP node, for a total of 640 to 19,200 requests per second for the entire 16 x 8 processor mesh. As mentioned in Section 5.2.4, however, the simulator throttles the request stream if too many requests are outstanding at any time. Thus, increased request rates given to the workload generator result in little increase in actual request rate, preventing infinitely long queues.

The curves in Figure 6-13 show how response time varies with both concurrency and file size. Simulations were run for file sizes of 4 KB to 32 KB; the graph does not show the 4 KB results since they are nearly identical to those for 8 KB files. For 32 KB files, performance begins to fall off at approximately 40 requests per node per second. This corresponds to a bandwidth of 1.2 MB/s per disk, which is comparable to the bandwidth achieved for individual large files. While transferring 40 32 KB files per node per second, response time is still under 100 ms from initial request until the time the last byte of file data is transferred. For smaller file requests, access latency begins to dominate. The maximum bandwidth for 8 KB files is

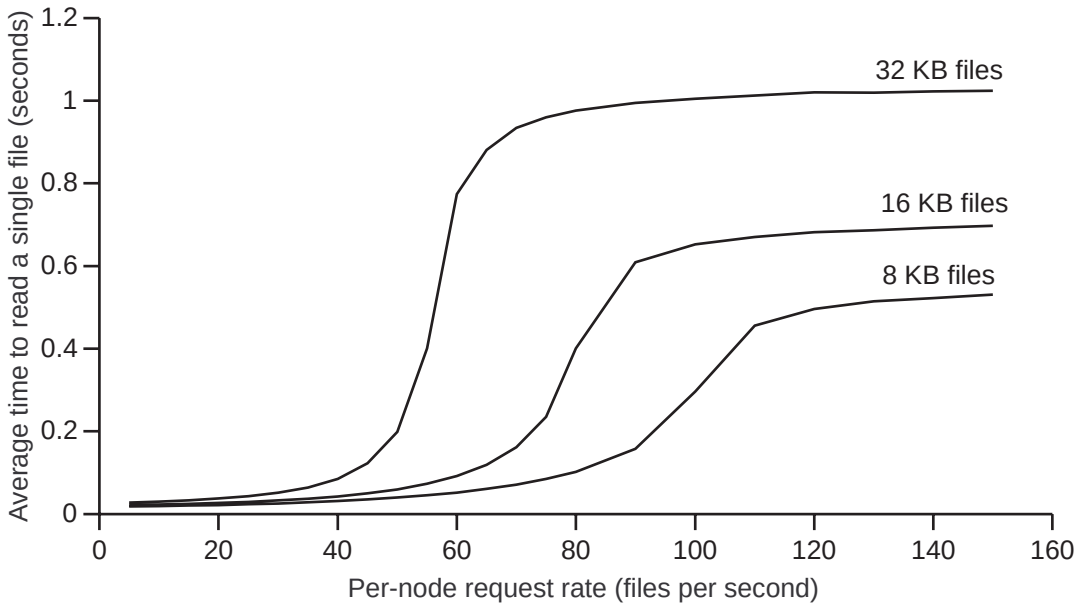

Figure 6-13. RAMA read performance for small files.

This figure shows RAMA's read performance for the small file workload described in Section 5.2.4 running on a 16×8 processor mesh. 75% of all requests were file reads, and only read performance is shown in this graph. RAMA performs very well at request rates of up 40 requests per second per MPP node. For smaller requests of 8 KB, 80 ms file reads are possible even at 60 requests per node per second. As expected, file read time drastically increases as the file system reaches its maximum throughput. At this point, additional delays in the request stream prevent request queues from growing infinitely large.

700 KB per second per node, though response time drops off drastically beyond 560 KB per second per node.

These performance figures show that the RAMA design does not sacrifice small file performance to gain good performance for large files. RAMA's design includes provisions for migrating metadata to tertiary storage, allowing the file system to store millions of infrequently used small files on tape without requiring disk to hold all of their metadata. However, the simulator does not model the additional disk access latencies due to metadata and directory lookup. These additional latencies are most severe for files on tape, since RAMA's storage scheme might require two or more tape reads to access a single small file on tape. In this section's simulations, however, all files were assumed to be on disk to gauge RAMA's maximum small file bandwidth. The simulator further assumes that the file system is simply given the correct bitfile ID for each file. RAMA would actually find a bitfile ID by hashing the directory name to find the directory, reading the directory to find the file name, and extracting the bitfile ID. Translating a file name to a bitfile ID would thus require approximately the same time as reading a small file. As a result, RAMA's actual performance on small files should be no worse than half that shown in Figure 6-13. Nonetheless, RAMA would still be capable of over 25 small file transfers per second per node with an acceptable response time. If directories are cached, RAMA's performance will improve because file name to bitfile ID translation would require no disk access.

6.4. Future Performance

The previous sections in this chapter have each varied a single parameter in the RAMA simulator to show their effects on overall system performance. While these simulations are helpful in understanding how individual parameters affect the RAMA file system, they also show that improving only a single parameter has a limited effect. For example, increasing linear density along a disk track provides gains at first, but it reaches a point where disk performance, and thus RAMA performance, is limited by other factors.

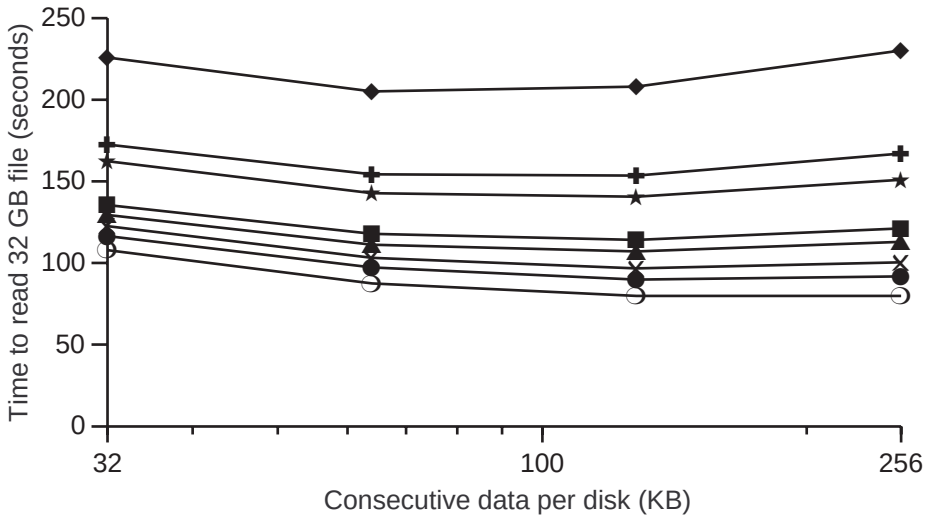

Figure 6-14. Projected RAMA performance using future technologies.

This graph projects RAMA's performance using faster disks than those available today. The top disk is the one used in most of the simulations, and uses data from the Seagate ST31200N disk. The bottom disk rotates at 9000 RPM, stores 144 KB per track, and has an average seek time of 4 ms and a maximum seek time of 9.1 ms. It has 4300 cylinders and 9 surfaces for a total capacity of 5.6 GB. The intermediate curves are for disks between the two extremes whose parameters are listed in Table 6-2. Each curve represents approximately a year of progress except for the bottom curve which is two years ahead of the one above it.

Each curve shows RAMA's expected performance for several different amounts of consecutive data per disk (as described in Section 6.2.1). Note that peak performance for current disks is at 32 - 64 KB per disk, while it increases to 256 KB per disk for the faster, denser disks of the future.

RAMA simulations using projected future disk capabilities are shown in Figure 6-14. The characteristics of the simulated future disks are shown in Table 6-2. The network used in all of these simulations is a 16 x 8 mesh running at 100 MB/s per link, as Section 6.1.1 showed that network performance was not a bottleneck. Two trends are evident from the data in the figure. First, RAMA performance improves as disk performance improves. Thus, RAMA should be able to take advantage of faster disks to provide higher bandwidth file service. Additionally, Figure 6-14 shows that the optimum amount of data to store on each disk increases as disk performance increases. File block size need not increase to accomplish this. Instead, RAMA can simply place more individual file blocks consecutively on a single disk before selecting a new disk. In this way, RAMA can increase its performance further using the high-performance disks of the future.

Capacity (MB)	Cylinders	Average track capacity (KB)	Seek time (ms)		Rotation rate (RPM)
			Average	Maximum	
864	2000	48	8	18	5400
1,108	2200	56	7.36	16.56	7200
1,394	2420	64	6.75	15.25	7200
1,724	2660	72	6.21	14	9000
2,106	2925	80	5.71	12.90	9000
2,782	3220	96	5.25	11.80	9000
3,568	3540	112	4.83	10.80	9000
5,573	4300	144	4	9.14	9000

Table 6-2. Parameters for simulated future disks.

This table lists the characteristics projected for future disks. The first disk is the Seagate 31200N, and the remainder are extrapolated using growth rates from [11]. RAMA's simulated performance using these disks in an MPP is shown in Figure 6-14.

6.5. Conclusions

The RAMA file system design scales well to future disk technologies and large processor arrays. While RAMA's performance is insensitive to network parameters such as bandwidth and latency, it does place a high load on slow networks. RAMA varies less than 1% in overall bandwidth between a 15 MB/s mesh and a 150 MB/s mesh, though the average load per link is ten times higher with the slower network. Since the network is not a bottleneck and the file system load is evenly and randomly distributed, the network topology — mesh, star, hypercube — has little effect on RAMA's simulated performance. For some network configurations such as a 3-D mesh or hypercube, RAMA performs well for several hundred nodes, as overall system bandwidth scales up linearly with the number of processor-disk pairs.

RAMA's performance is disk-limited; thus, improving disk parameters such as seek time, linear data density along a track, and rotation rate result in higher file system performance. However, improving only one disk characteristic while fixing the others result in diminishing returns. Initially, RAMA performs significantly better. As the fixed components dominate the time needed for a single I/O, though, performance flattens out. Of the three disk parameters considered, rotation rate is the most important as it both decreases rotational latency and increases sustained transfer rate. Quadrupling rotation rate while leaving the other two parameters at current values results in more than double the bandwidth. Denser disks also improve performance, but require the file system to store more data per disk to take full advantage of them. If current disks only packed data denser, overall RAMA bandwidth would increase by less than a factor of 2 even as density increased by a factor of 6 if RAMA did not store more data per disk. Faster access times, likewise, increase the file system bandwidth. However, their influence is even smaller, as reducing the average seek time to 4 ms produces, at best, a 20% improvement. The high concurrency at each disk and the request ordering algorithm ensure that seeks are short, so improvements in average and maximum seek time have little effect.

RAMA has few design parameters to vary — indeed, one of its major advantages is its simplicity. Using more than one disk per MPP node produces, as expected, higher performance. For relatively few disks per node, performance scales linearly. As more disks are added, however, performance drops away from linearity. This effect is caused by insufficient concurrency in the simulated request stream. Good performance in RAMA depends on having many requests outstanding at any given time; failing to do so prevents linear

scaling. The same problem occurs when the amount of data stored on each disk is increased. Performance can vary by as much as 15%, while the optimum amount to store on each disk increases as disk performance increases.

While RAMA is designed primarily for large files, it performs adequately for small files as well. As a result, RAMA is well-suited to networked environments including both MPPs and workstations. While MPP files are often megabytes or gigabytes in length, workstation files tend to be considerably smaller — well under 1 MB long. RAMA's performance on a workload composed of small file transfers shows that it can serve small requests from workstations as well as large requests from MPP applications. This allows workstations to manipulate files on an MPP without having to copy them to a workstation-based file system.

This chapter has detailed the sensitivity of the RAMA design's performance to technological and design parameters as well as the size of transferred files. The following chapter fixes these parameters, and compares RAMA's performance to that of standard striping. As the next chapter shows, RAMA provides a high-performance, configuration-free alternative to conventional striped MPP file systems.

7 A Performance Comparison of RAMA and Striped File Systems

This chapter discusses the performance of the RAMA file system for several massively parallel applications. It demonstrates that using hashing to distribute file system data among all the disks in a massively parallel processor (MPP) performs well relative to traditional striped file systems.

Traditional striped file systems for MPPs use dedicated I/O nodes, each of which has one or more disks attached to it. Such a file system has many configuration options, all of which can drastically affect application performance. These variables include the number of I/O nodes, the number of disks per I/O node, and the stripe unit size both within an I/O node (if it contains more than one disk) and across all of the I/O nodes in a stripe. Many current striped file systems allow a programmer to control some or all of these options to provide her application with maximum file system bandwidth to a single file. However, not all options can be reset by the application — a program running on a system with 8 I/O nodes cannot request to stripe data across 16 nodes. Worse, programs that specify file striping are often difficult to port, as the system's I/O configuration is likely to change even between different machines of the same model.

RAMA alleviates these problems by providing excellent performance without requiring extensive user configuration. Instead, every program benefits from high bandwidth access to its files without having to specify how each file is striped. This allows programmers to be concerned more with making an algorithm work and less with how to deal with the many possible I/O node configurations.

The chapter first shows that distributing data pseudo-randomly among all of the disks in an MPP, as RAMA does, imposes little penalty over optimal data striping on disk. Moreover, the pseudo-random data placement of RAMA performs significantly better than non-optimal striping. As a result, RAMA is preferable for use when the application does not explicitly specify how data should be stored on disk. RAMA's worst-case performance is considerably better than that of a striped file system, and it is only a few percent worse than a striped file system in the best case.

Another concern with RAMA is that the movement of file system data between nodes of an MPP creates network congestion because the data may not be on a disk "near" the processor that needs it. As this chapter shows, RAMA consumes little network bandwidth on a modern MPP with a high-speed interconnect. Additionally, it uniformly spreads its generated traffic over the entire machine. In contrast, a standard striped file system concentrates its network traffic near specific I/O nodes, causing congestion by concentrating all I/O requests on just a few nodes' links. Both file systems must send the same number of requests to the disks; however, RAMA distributes them more evenly in the network than does a striped file system.

7.1. Application Performance

7.1.1. Strided Access

The first set of simulations comparing RAMA and a standard striped file system used synthetic workloads. These workloads consisted of two types — those that represented real applications that used strided access, and those that simply read or wrote the file as rapidly as possible.

7.1.1.1. Raw Bandwidth Experiments

In the raw bandwidth tests, each node in the parallel processor read or wrote a single chunk of contiguous data and waited for every node to complete its corresponding operation. This process was repeated until the entire file was transferred. The nodes did not do any computation on the data; they merely attempted to move data between disk and memory as quickly as possible.

The order in which the nodes read the data had a major effect on the bandwidth delivered by the striped file systems. The experiments covered two transfer orderings — iteration sequential and node sequential, analogous to row-major and column-major array element orderings. Figure 7-1 graphically shows the access

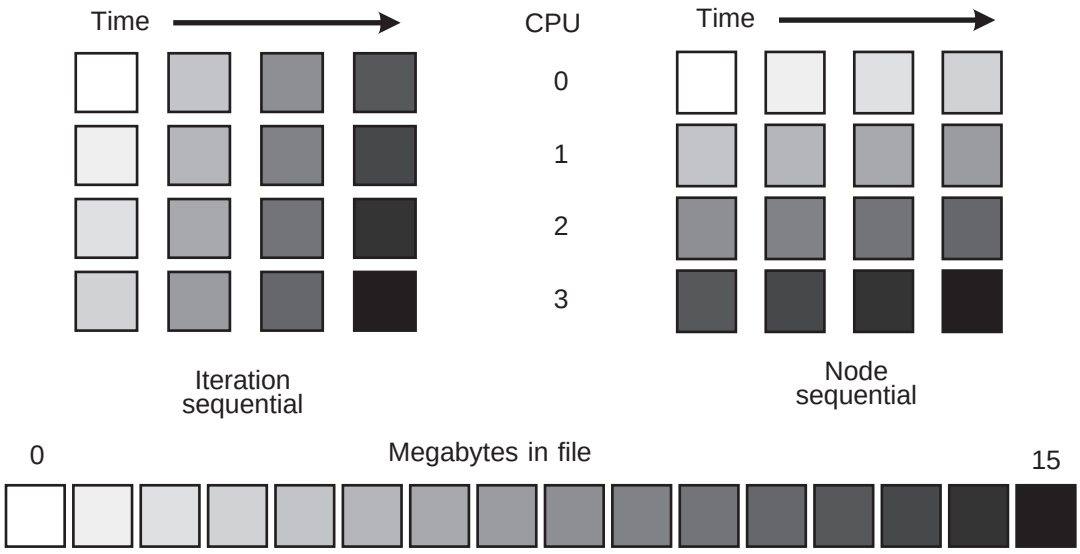

Figure 7-1. Node sequential and iteration sequential request ordering.

This diagram shows the two access patterns used in the experiments that transferred an entire file. These patterns correspond to row-major and column-major accesses in a program that uses matrices. An iteration-sequential requests one contiguous chunk of the file in each iteration of the main loop. However, individual nodes of the MPP do not transfer data sequentially between iterations. Node-sequential access exhibits the opposite characteristics: a node's I/O is sequential between iterations, but the data moved during each iteration is not contiguous in the file.

patterns for the two orderings. In iteration sequential accesses, transfers appear sequential between iterations. Taken as a whole, the MPP transfers data sequentially. However, each individual node does not appear to have sequential transfers; instead, the offsets of consecutive accesses into the file differ by the total data transferred by all nodes during each iteration. For the node sequential access pattern, on the other hand, each individual node makes all of its requests sequentially. When viewed from the total machine level,

however, the accesses do not appear sequential. Instead, transfers by adjacent nodes in the MPP start at offsets differing by *fileSize/nodes*.

The access pattern of the whole-file transfer had a major effect on the overall bandwidth of the striped file system, but had little effect on the performance of the RAMA file system. For a striped file system, bandwidth varied by a factor of five between the best and the worst striping arrangements, as Figure 7-2 shows. The difference in performance is caused by hot spots in the disk array. Figure 7-3 shows the problem in detail for the node sequential access pattern. For very small stripe sizes, any request will access many disks, achieving good bandwidth for each individual request. Hot spots are minimized, since every node accesses many disks. As the stripe size increases, each node accesses fewer and fewer disks to complete its request. If different nodes use disjoint subsets of the disks available, performance remains good. However, overall bandwidth drops sharply if all of the nodes direct their requests to only a few of the available disks. When each node is requesting a portion of the file on the same disk, medium-sized stripe sizes are a very poor choice. As the file system's stripe size increases, however, performance goes back up. This return to good performance occurs because each node effectively gets its own disk for all of its requests. While overall performance is good under this scheme, each node performs relatively poorly for its accesses. The request from any node is only served by a single disk, limiting its performance to that of one disk.

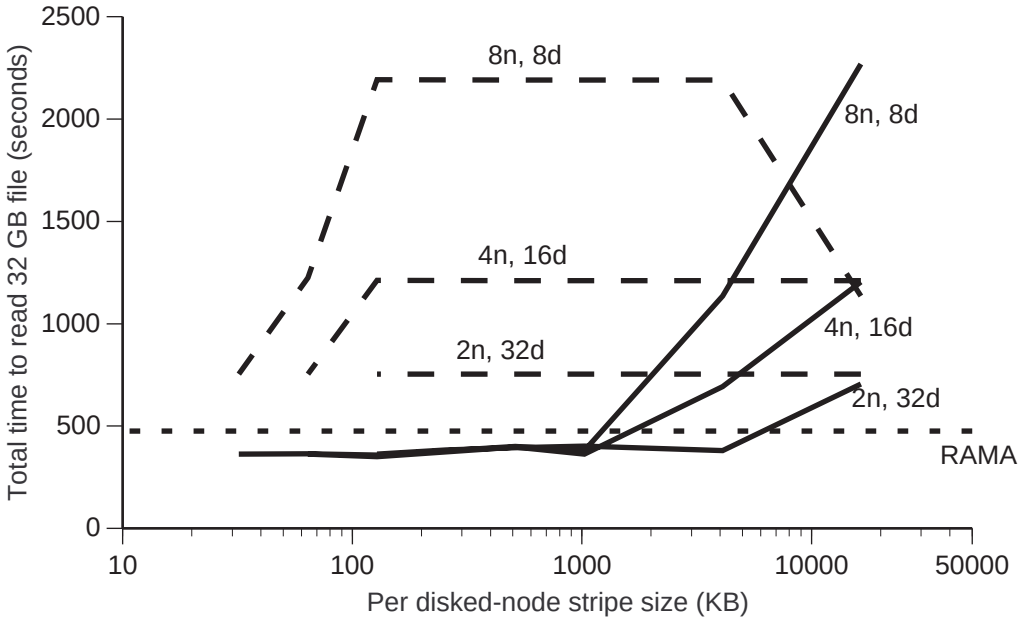

Figure 7-2. Comparison of raw bandwidth in RAMA and striped file systems.

This graph compares the performance of the RAMA design to that of striped file systems in a test of raw bandwidth. The data is read in two different access patterns as described in Section 7.1.1.1. The iteration-sequential reads are shown as the solid lines, and the dotted lines represent node-sequential reads. Each line is labeled with two numbers: *a*n, *b*d. *a* is the number of disked nodes for that series of tests, and *b* is the number of disks attached to each disked node. There is only one line for RAMA, as its performance varied by less than 0.4% between the two workloads. Additionally, stripe size is not a parameter to the RAMA file system; thus, the line for RAMA's performance is based on just a single set of file system parameters.

Under the RAMA file system, though, there is no stripe size. Instead, file system requests are broken into chunks and hashed to various nodes in the file system. Effectively, a file is "striped" across all of the disks

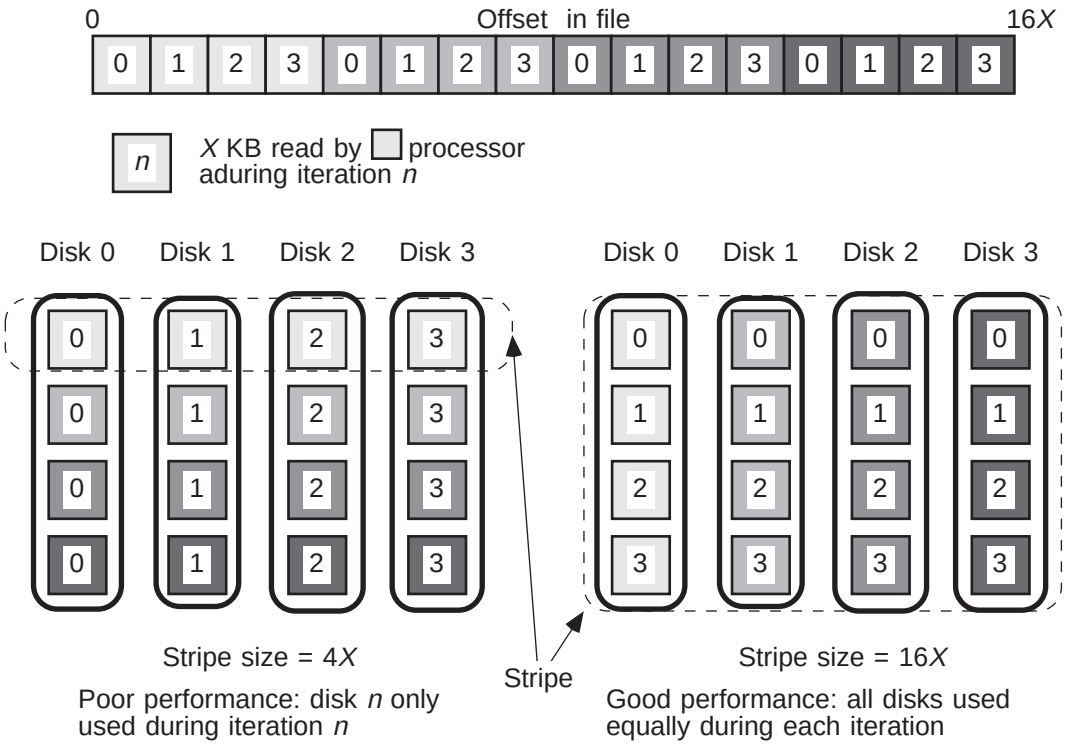

Figure 7-3. Interaction of stripe size and application stride.

This figure shows the interaction between file system stripe size and application stride when transferring a file to disk. It shows that an application achieves maximum bandwidth to the file system when its access pattern keeps all of the disks busy all of the time, as on the right. The program on the left evenly stores its data on the disks; however, it only uses one disk at a time, limiting the performance of the four disk system to the bandwidth of a single disk.

in the processor array in a pattern that no application is likely to duplicate. Figure 7-2 shows that bandwidth to the file system is comparable to the best performance from a striped file system.

Note that RAMA's peak performance is lower than that of a well-tuned striped file system. For the test in Figure 7-2, RAMA takes 10% longer to read the 32 GB file than the optimally striped file system. This performance loss occurs because, if the stripe size is chosen properly, requests will be uniformly distributed across all the disks in the system at all times. Since all disk accesses in an iteration must complete before the next iteration begins, each iteration will take as long as the longest time spent by a single disk. For a striped file system under ideal conditions all disks will finish close to the same time, since they are all servicing the same number of requests.

In RAMA, however, the total number of requests each disk must service is not distributed uniformly, but instead multinomially. The average number of requests per disk remains the same as in the striped file system, but the maximum number of requests handled by any single disk is higher than in the ideal striping case. As a result, each iteration takes slightly longer to complete because it waits for the disk with the highest load to finish servicing its requests.

The difference between RAMA's performance and that of the ideal striped file system depends on several parameters: the number of disks and processors in the system, the total amount of data requested each iter-

ation, and the average time to service a single request. Note that the last parameter's value might actually differ between RAMA and a striped file system because of differences in seek time.

7.1.1.2. Strided Access with Computation

Many programs step through a large data set once per iteration. Such applications fall into two groups. The first are those that only need to write out their results after each iteration. Their data fits entirely in memory, so the application does not need to use the disk as temporary storage. Instead, it uses the disk to store a history of the model's state for later analysis. Other programs read many gigabytes off disk, analyzing them and writing a relatively small output file at the end. The access patterns for both of these applications can be described with one model, as both do regular sequential I/O interspersed with computation. Out-of-core algorithms, such as matrix decomposition described in Section 7.1.2, perform both reads and writes each iteration. These cannot be described with the simple model used to specify I/O patterns for the experiments in this section.

The next set of experiments use a synthetic workload that overlaps I/O with computation. Asynchronous file system requests allow an application to explicitly make read-ahead and write-behind requests. Instead of an application requiring the *sum* of the time spent computing and doing I/O, it only requires the *maximum* of the two. For some applications, CPU time dominates I/O time. These applications perform similarly regardless of the file system, since they are CPU bound. Other applications have I/O requirements that meet or exceed the time necessary for computation, making the programs I/O bound. The experiments in this section deal only with I/O bound models since CPU bound applications will not benefit or suffer from differing file system performance.

The performance of a write-dominated strided access application is shown in Figure 7-4. This program uses 100 MFLOPs per iteration for each megabyte of data written. This corresponds to 800 operations for each double-precision number written. On a parallel processor with 100 MFLOPS processors, the application would write out 1 megabyte per second per node. If the disk system can keep up with this I/O rate, the application will be CPU bound. As Figure 7-4 shows, RAMA can provide sufficient bandwidth to allow nearly 100% CPU utilization for all of the data layouts. However, striped file systems can only sink 1 MB/s if the data layout "fits" the striping arrangement. To achieve maximum performance from striping, an application programmer must know about the underlying disk configuration in the MPP. If the placement knowledge is missing or incorrect, the program can run 8 times slower than RAMA. Even if striping is done properly, RAMA is within 5% of the performance of a striped file system. RAMA's ability to provide good file system performance without configuration is one of its major advantages over conventional MPP file systems.

Simulation results for a read-dominated program are similar to those for write-domination, as the simulator's file system model does not account for metadata. Since the simulator does not do extra I/O for metadata, two programs differing only in whether data is read or written would perform nearly identically.

Section 7.1.1.1 shows that the file system is capable of slightly more than 1 megabyte per second per disk. The previous example required exactly 1 megabyte per second per disk, so it is on the verge of being I/O bound. If a program requires more disk bandwidth, it will be I/O bound unless the disk system is sped up. This can be done in one of two ways: increase the bandwidth of individual disks, or use additional disks. Both of these options were discussed in Chapter 6.

7.1.2. Matrix Decomposition

The second group of experiments involves matrix decomposition as described in Section 5.2.2. The decomposition program, like the other applications in this chapter, uses asynchronous I/O to hide much of the latency in the file system. Thus, the file system needs merely to provide I/O sufficiently fast to prevent the

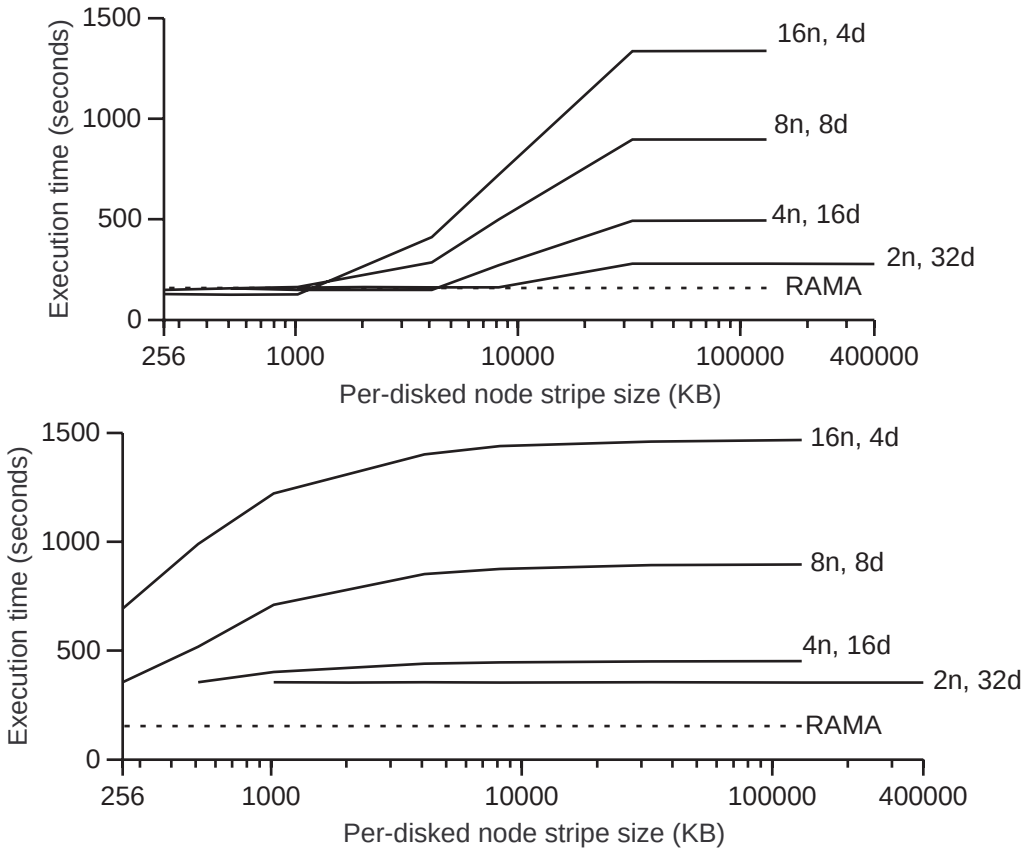

Figure 7-4. Performance of a synthetic workload doing write-dominated strided access under RAMA and striped file systems.

The two graphs in this figure show the performance of synthetic workloads that model applications doing strided access on a 16 x 16 processor array under both RAMA and various configurations of striped file systems. Both applications average 1 MB/s of I/O per node, and overlap I/O with computation using write-behind. The top graph shows an application doing iteration sequential access, where small stripe sizes perform best. RAMA is within 5% of the performance of the best striping arrangement. The bottom graph shows a program with node sequential access where no striping arrangement outperforms RAMA.

program from becoming I/O bound. The point at which this occurs depends on the file system and several other factors — the number and speed of the processors in the MPP, and the amount of memory the decomposition is allowed to use.

Figure 7-5 graphs the total amount of I/O done by the decomposition of matrices of various sizes against the memory space used by the algorithm. Note that, even with a nearly infinite amount of memory, the program must perform some I/O to read the matrix initially and write out the result. The graph shows the average bandwidth for a decomposition running on an MPP with 100 MFLOPS processors; for faster or slower MPPs, the bandwidth must be scaled by the ratio of the actual speed to 100 MLOPS.

Because matrix decomposition uses asynchronous I/O, the application will only run slower if the file system is unable to keep up with the CPU. Clearly, if the average required bandwidth is more than the file system can deliver, the application will be I/O bound. However, an application may be I/O bound for only part of its execution time. This is particularly likely to happen with a striped file system with a poorly selected stripe size. Matrix decomposition stresses striped file systems by only transferring a fraction of the file each

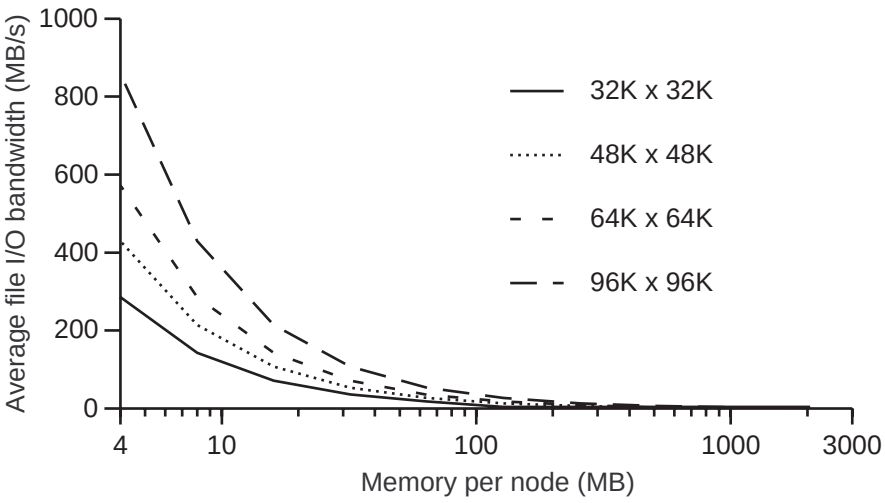

Figure 7-5. I/O done by matrix decomposition on a 64 processor MPP.

If insufficient memory is allocated to the matrix decomposition algorithm, it requires more I/O than a file system can reasonably provide. However, the I/O demand becomes reasonable even for memory sizes as small as several megabytes. As the algorithm uses more memory, the I/O required drops off sharply. While the average I/O bandwidth never reaches 0, it does drop below 5 MB/s for very large memories. This graph shows the average I/O bandwidth used matrix decompositions on an 8 x 8 processor mesh with 100 MFLOPS processors.

iteration, resulting in the performance shown in Figure 7-6. Under RAMA, the performance of matrix decomposition matches that of the best striped arrangements, and is far better than the worst arrangements. Note, too, that the best performance for the striped file system occurs with small stripes.

Just a small change in the source code governing the placement of data, however, can cause a large difference in performance for LU decomposition under striping. The data in Figure 7-7 were gathered from a simulation of a nearly identical program to that from Figure 7-6. The sole difference was a single line of code determining the start of a segment of the matrix. An arbitrary choice such as this should not cause radical changes in performance. Under RAMA, the application's execution time is the same to within less than 0.1%. Performance under file striping, however, is very different for the two data layouts. The small stripe sizes that did well in the first case now perform poorly with the alternate data layout. On the other hand, large stripe sizes serve the second case well, in contrast to their performance with the first data layout.

7.1.3. Global Climate Modeling

The global climate model consists of two separate applications: one that models the physics and dynamics of the major components of the atmosphere, and another that uses the output of the first model to compute the behavior of many chemical species in the atmosphere. In one scenario for running these two programs, all of the output from the first model is saved on disk and read back later by the second model. The I/O for these programs is complicated because the two applications prefer different data organizations.

The GATOR model, as described in Section 5.2.3, simulates the behavior of various chemical species in the atmosphere. It takes its input from the checkpoints of a standard global climate model. The standard GCM stores its data in longitudinal strips, each stretching from near the north pole to near the south pole. GATOR, however, wants to group data into blocks, consisting of small chunks of adjacent longitudinal strips as

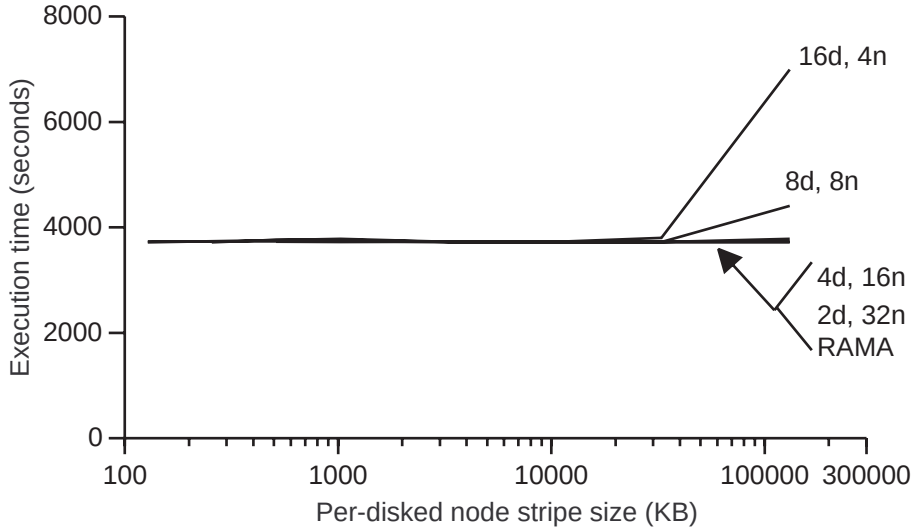

Figure 7-6. Execution time for LU decomposition under RAMA and striped file systems.

LU decomposition suffers only a 2% performance penalty for running under RAMA rather than a striped file system with a small stripe size. RAMA outperforms striping with large file sizes by a factor of 3 in the worst case. For this application, stripe size has a strong effect on application performance. RAMA provides nearly equivalent performance without requiring the user to configure the file system.

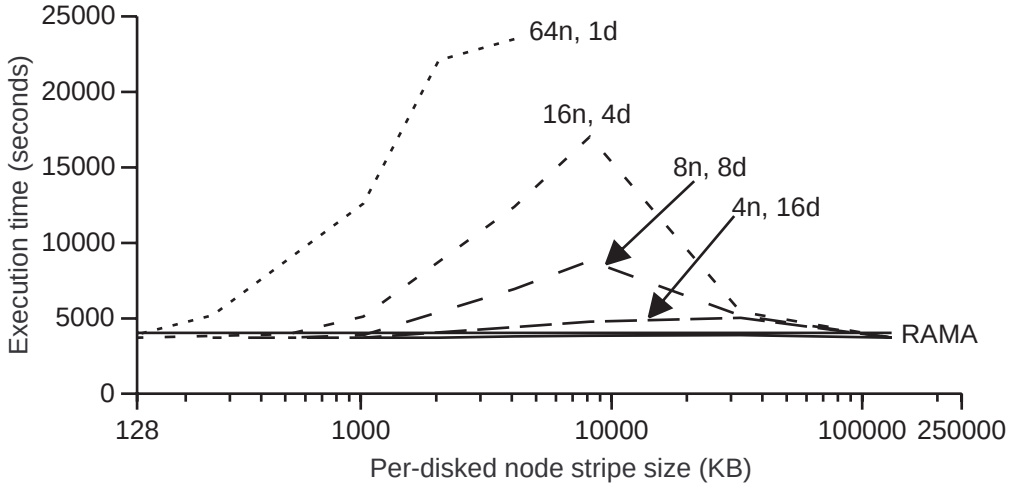

Figure 7-7. Execution time for LU decomposition with an alternate data layout.

This graph shows the same program and file systems as Figure 7-6 with just a single change to the application's code that determines how data is laid out on disk — similarly to either row order or column order. The small change results in radically different performance under striping, but execution time under RAMA is the same in both cases.

shown in Figure 5-9. This decision forces GATOR to make many relatively small requests to read the data for each iteration.

The simulated GATOR model uses a 144 longitude by 96 latitude grid. Each grid point has 8 KB of data to represent the current atmosphere at 20 levels. This information is read in every 5 simulated minutes. The model executes 0.8 MFLOPs per grid cell per iteration, so the system needs to read in 114 MB of data for each 11,000 MFLOPs. Since the simulated system has 100 MFLOPS per processor, the total necessary I/O bandwidth is about 1 MB per second per CPU. This matches well with the requirement for one disk per CPU.

Striped file systems generally perform well handling GATOR's file requests. While each individual node does not make sequential requests, the overall request pattern is similar to the iteration-sequential pattern mentioned in Section 7.1.1. During each iteration, GATOR reads 114 MB. Striping attained maximum bandwidth whenever a stripe across the entire file system evenly divided 114 MB — i.e., when a single iteration's request consisted of a whole number of full stripes across the file system. Under these conditions, the file system mapped the same number of requests to each disk in the MPP, giving maximum performance as shown in Figure 7-8.

However, GATOR's performance declines markedly if the file system uses large stripes. The decline occurs because some disks must service more requests than others. This effect mirrors that seen in the iteration-sequential I/O case for large stripe sizes. At any given iteration, the distribution of file requests to disks is uneven, even though the file as a whole is evenly distributed among the disks in the file system.

GATOR's performance on RAMA varies with the size of the processor mesh. For meshes up to 8 x 8, GATOR using RAMA performs as well as the best striped file system. For larger meshes, GATOR's performance under RAMA declines. Using an MPP with 128 processors, GATOR loses 40% in performance between an optimally-striped file system and RAMA.

The drop in performance on larger MPPs occurs because GATOR does not scale its problem size with the size of the MPP it runs on. Since the problem size is constant, the average number of outstanding I/Os per disk declines as the MPP becomes larger. For an optimally striped file system, this does not present a problem, as each disk services the same number of I/Os. RAMA, however, distributes data pseudo-randomly to disk. As the mean number of requests per disk declines, the variance increases. This results in a greater variation between the average number of requests per disk and the number that the most heavily loaded disk must satisfy. The ratio between maximum number of requests per disk and mean requests per disk determines the difference in performance between optimal striping and RAMA — as the ratio increases, RAMA's performance decreases relative to striping.

GATOR thus demonstrates that RAMA performs best when the application has many outstanding requests for data. If the concurrency drops too low, RAMA will perform worse because of its pseudo-random distribution. For many applications, however, this is not an issue. Given a larger and faster MPP, scientists usually increase the size and detail of the simulation rather than solving the original problem faster.

7.2. Disk Utilization

Application execution time and peak bandwidth are only two aspects of performance that are important to system designers. Disk utilization is another major performance metric for file systems. For parallel file systems, in particular, the spatial and temporal distribution of requests to disk affect overall file system performance. In addition, the distribution of file data to disk affects storage utilization. If one disk receives more data than other disks, the file system may seem full even though there is remaining free space.

7.2.1. Spatial Distribution of Disk Requests

Spatial distribution of requests refers to the overall request load on each disk in the file system. Over the long term, this each disk in the file system should have the same utilization for optimal performance. Underutilized disks represent unused bandwidth, and reduce the overall performance of a file system.

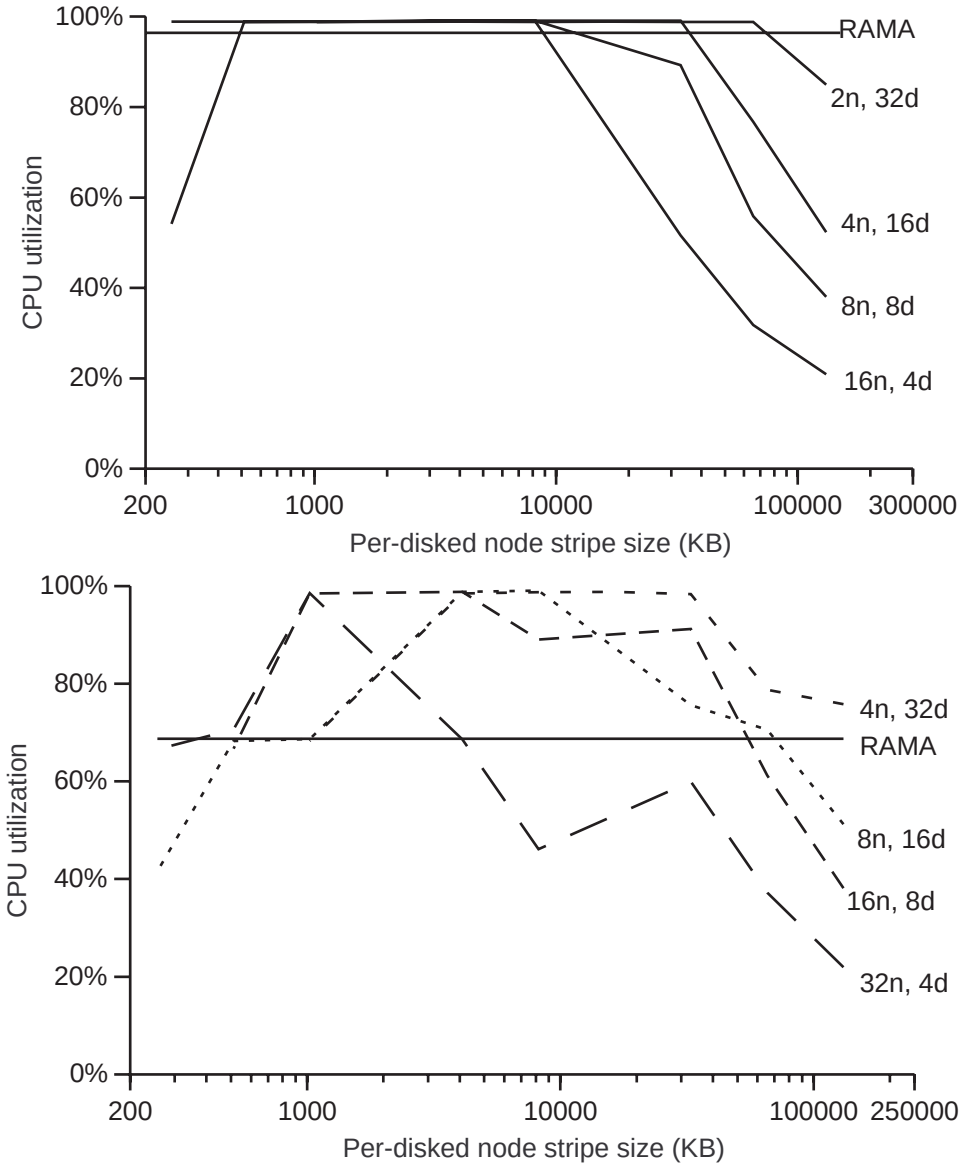

Figure 7-8. GATOR performance under striping and RAMA.

The two graphs in this diagram show the performance of GATOR under striping and RAMA. The top graph shows the problem running on an 8 x 8 processor mesh with 64 disks, and the bottom shows a 16 x 8 processor mesh.

For an 8 x 8 processor mesh, GATOR running under RAMA performs almost as well as under striped file systems, losing only 2% processor utilization. With larger stripe sizes, standard file systems' performance falls off, dropping to CPU utilization levels as low as 21%. However, the 16 x 8 case is not so rosy for RAMA. Here, the low average number of I/Os per disk leads to a higher variance of requests per disk. Since RAMA's performance is limited by the disk with the most I/Os, GATOR runs at only 68% processor utilization in this case.

In a striped file system, read and write requests are mapped to disks in an regular pattern. This order allows a striped file system to guarantee that sequential read and write requests to a large file will be evenly dis-

tributed among all disks, as shown in Figure 7-9. If the entire file is read or written, every disk handles the same number of requests. If the requests for each disk are also temporally distributed, as described in Section 7.2.2, the file transfer will attain maximum bandwidth.

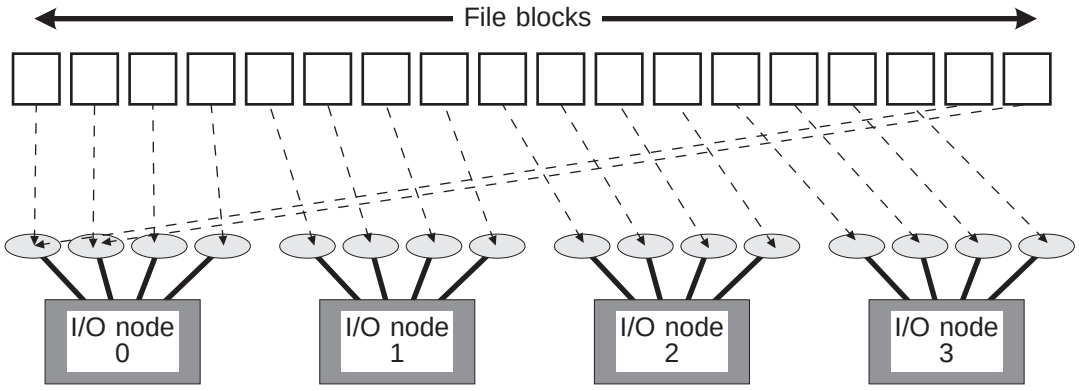

Figure 7-9. Sequential request distribution in a striped file system.

In a striped file system, data is distributed evenly to each disk. The regular pattern guarantees that each disk stores the same amount of a file's data. However, the tail file does not occupy an integral number of stripes. As a result, the first two disks of I/O node 0 will service twice as many requests as the rest of the disks if the entire file is transferred sequentially.

The biggest problem with disk request distribution in striped file systems occurs when file requests only cover a portion of a full stripe. This is a real issue in systems with hundreds of disks. Even if each disk holds just 4 KB of data, a full stripe in a system with 256 disks will cover 1 MB. Many striped file systems, however, place more data on each disk, making a full stripe several megabytes in length.

If all request sequences start with the same disk, the disks towards the end of the stripe will service fewer requests than the disks near the start of the stripe. Some file systems accept this penalty. Other MPP striped file systems, such as Vesta [18,19], allow each individual file's stripe to start on a different disk. In this way, long-term distribution of requests to disks remains uniform even when files do not use an entire stripe.

Figure 7-10 shows an example of the partial stripe access problem, as typified by matrix decomposition. While the LU decomposition algorithm writes each byte in the file exactly once, it does not distribute its reads evenly. Only the data below the diagonal of the matrix is read multiple times. As a result, requests are not evenly distributed to disks. In Figure 7-10, the variation between the disk with the most requests and the one with the fewest is a factor of 2.5. This disparity limits the performance of the file system. Starting each file's stripe at a different disk would not help because the decomposition algorithm only references a single file. This distribution problem can only be solved by picking the "correct" stripe size, which is dependent both on the application's access pattern and the number and arrangement of the disks in the system.

RAMA, on the other hand, uses a hashing algorithm to pseudo-randomly map file system requests to disks. This operation converts any regular input ordering into a random-looking output ordering. Mapping requests to disks this way leaves the question of whether the request load will be evenly distributed among all of the disks in the file system.

Figure 7-11 graphically shows the distribution of disk requests processed during the read of a 32 GB file on both a 64 and 256 processor MPP running RAMA. The disks are labeled starting at 0, with one for each processor. As the diagram shows, both the 64 and 256 processor cases exhibit good distribution, as all of

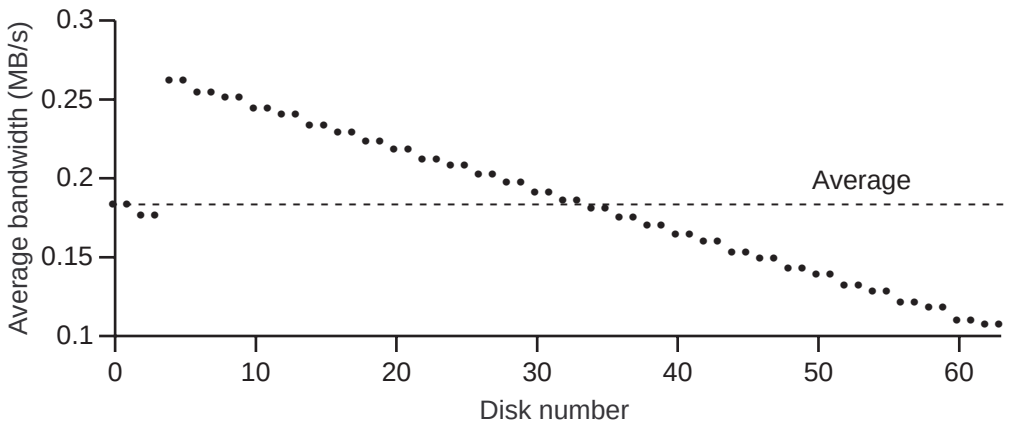

Figure 7-10. Poor distribution of requests to striped disks for LU decomposition.

When application access patterns interact poorly with file stripe sizes, the request distribution across all of the disks in a striped file system can vary greatly, as shown in this graph of the disk load during the decomposition of a 32,768 x 32,768 matrix. This graph shows the average bandwidth delivered by each of 64 disks in a striped file system, clustered 4 disks per I/O node over 16 I/O nodes. The most heavily-loaded disks averaged 0.26 MB/s, while the least-used disks averaged just over 0.1 MB/s. Since the program used the file system inefficiently, its execution time was 4 times that of the most efficient striping arrangements. This long execution time lowered the disks' average bandwidths, since the total amount of data transferred is constant for all choices of stripe size.

the disks experience approximately the same request load. This even distribution is important because, as noted in Section 7.1.1.1, the performance of the file system is limited by the disk that must service the most requests.

Unlike many striped file systems, RAMA does not need application hints to avoid the problem of partial stripe transfers. RAMA stores a fixed quantity of data on each disk before rehashing the <*fileId, blockOffset*> pair and switching to a new disk, as described in Section 4.1. Since blocks for a file are distributed pseudo-randomly, any request that spans multiple disks will access pseudo-randomly chosen disks. The hashing algorithm makes it unlikely that any two specific subsets will be identical. This effect can be seen in Figure 7-12, which shows the distribution of requests to disks for LU decomposition. This algorithm does not read the entire file each iteration. Rather, it reads only the portion of each column under the diagonal during the update stage. This amount varies with each column, and is often smaller than a single stripe would be. As Figure 7-12 indicates, however, requests are still evenly distributed among disks.

7.2.2. Temporal Distribution of Disk Requests

For a file system to perform well, requests to its disks must be distributed uniformly in time as well as in space. Even if a system maps the same number of requests to each disk over the course of a program's run, performance will suffer if individual disks' requests are clustered together.

This effect often appears with striped file systems. Section 7.1.1.1 describes a case where a different access ordering results in a large performance loss. Since the same data is being read, spatial distribution between the two orderings is identical. However, temporal distribution differs greatly between the two access patterns. The same regularity that guarantees uniform distribution of requests to disks also makes it possible for a program to cluster many requests to a specific disk within a short period of time. In an MPP, though, the problem is complicated because many processors may each have their own reference stream. While any one processor's stream may stress only a few disks, all of the streams proceeding at the same time can cover

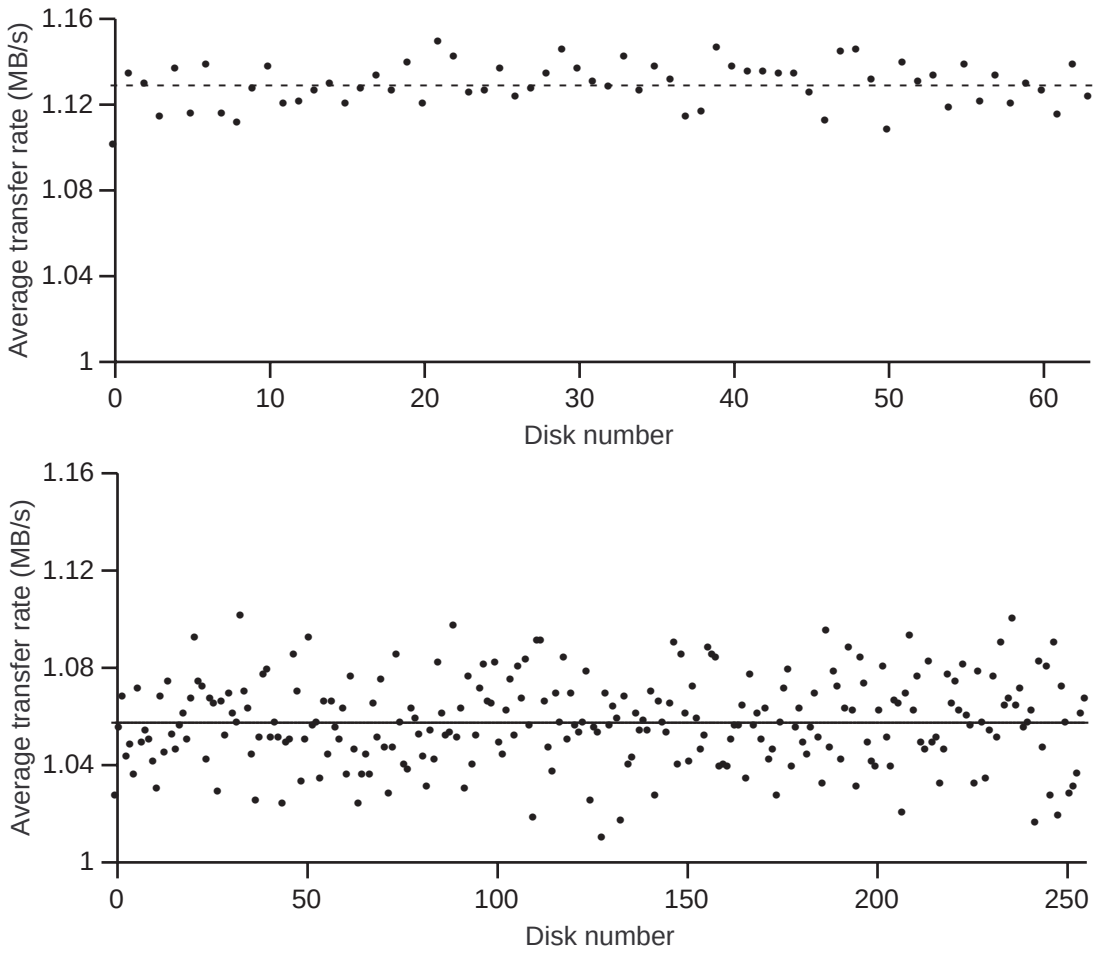

Figure 7-11. Distribution of requests to RAMA disks during the read of a 32 GB file.

The scatter plots in each of these graphs show the average bandwidth each disk provided during the read of a 32 GB file. The top graph shows the distribution for an 8×8 mesh, while the bottom graph is for a 16×16 mesh. Since all requests were 32 KB long, the number of requests serviced is proportional to the bandwidth. The difference between the disk with the lowest average transfer rate and the horizontal line corresponding to the average bandwidth per disk is the loss in performance suffered between the ideal striping case and RAMA.

all of the disks. Figure 7-13 shows both a well-performing and poorly-performing temporal distribution. In the well-performing case, each processor in the MPP touches half of the disks during each iteration. However, all of the disks are in use at any given time. The poorly-performing case, on the other hand, has every processor's requests going to the same set of disks each iteration. The disks vary by iteration, but performance suffers because half of the disks are idle at any given time. The loss in performance depends on the fraction of disks idle each iteration, but drops of a factor of 8 or more can occur for poor striping configurations.

Because RAMA distributes data pseudo-randomly, it is largely immune to poor temporal distribution of requests to disks for the same reasons that it is relatively immune to poor spatial distribution. Any regularity in time is randomized by the hash function, so the distribution of requests at any instant appears random. Figure 7-14 shows how effective this redistribution is, using the same workload shown in Figure 7-13.

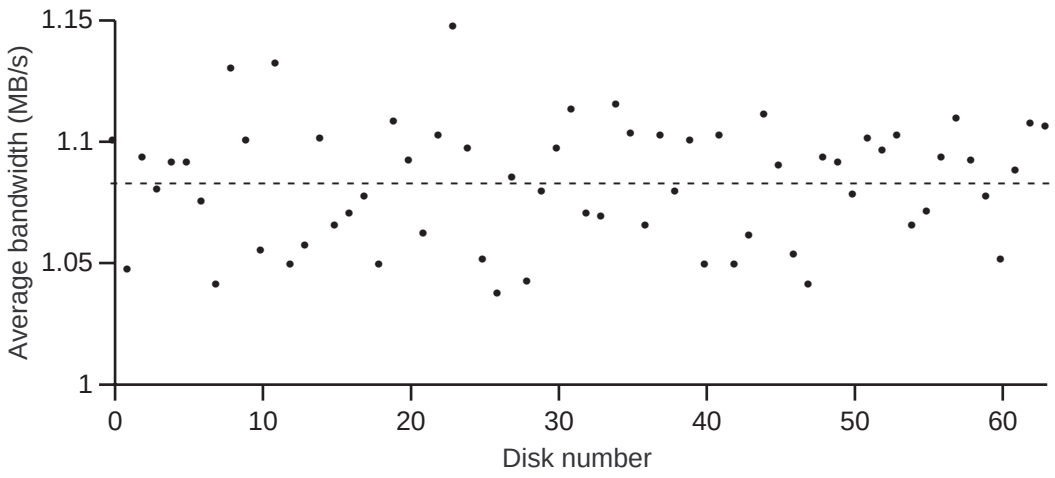

Figure 7-12. Distribution of requests to RAMA disks for LU decomposition.

This graph, like Figure 7-11, shows the average bandwidth provided by each disk in an 8 x 8 processor array running RAMA. In this figure, the MPP is doing a 24,576 x 24,576 matrix decomposition with segments 512 elements wide. Again, the spread between the disk with the fewest requests and disk with the most is requests is small. Here, the least-used disk averages 1.03 MB/s and the most-used 1.15 MB/s, so the spread is 11% of the average bandwidth. No disk is more than 6.5% over or 4.6% under the average bandwidth per disk.

While the pattern of disk accesses for striping is regular in both cases, RAMA scatters the disk requests for both access patterns. This results in slightly lower performance in the optimal case because of a slightly non-uniform request distribution in RAMA, similar to that described in Section 7.2.1. However, performance in the non-optimal case is far superior for RAMA — it suffers no degradation between the two cases, while the striped file system loses a factor of 8 in performance.

7.2.3. File System Data Distribution

Another issue file systems must face is that of file block allocation. Poor allocation schemes result in several problems including inefficient storage usage and poor performance from excessive disk seeks. RAMA's disk block allocation scheme and the use of tertiary storage as a backing store address both of these problems.

A typical striped file system may allocate all of the blocks on its disks because there is no intrinsic location for any file block. These system use a combination of simple mapping and index lookups to find a block. For example, a simple Unix-based file system using a disk array would look up a logical disk block number using standard Unix file indices, and map that logical disk block to a physical disk block with a few simple arithmetic operations. Vesta [18,19], on the other hand, uses simple operations to compute the disk particular data lies on, and then does an index lookup using indices local to the disk.

Since standard and striped file systems use indices to locate every block, individual blocks are limited to, at worst, a single disk. In addition, these file systems use striping to ensure that data is distributed evenly among all of the disks in the file system. Theoretically, then, standard file systems can use all of the available storage space for data. However, many file systems reserve free space to make disk block allocation faster. The Unix Fast File System (FFS) [54] typically reserves 10% of a disk to keep acceptable performance. LFS [74], on the other hand, needs closer to 20% of the disk free to maintain acceptable performance. Many parallel file systems are based on one of these two models. For example, the Vesta file system

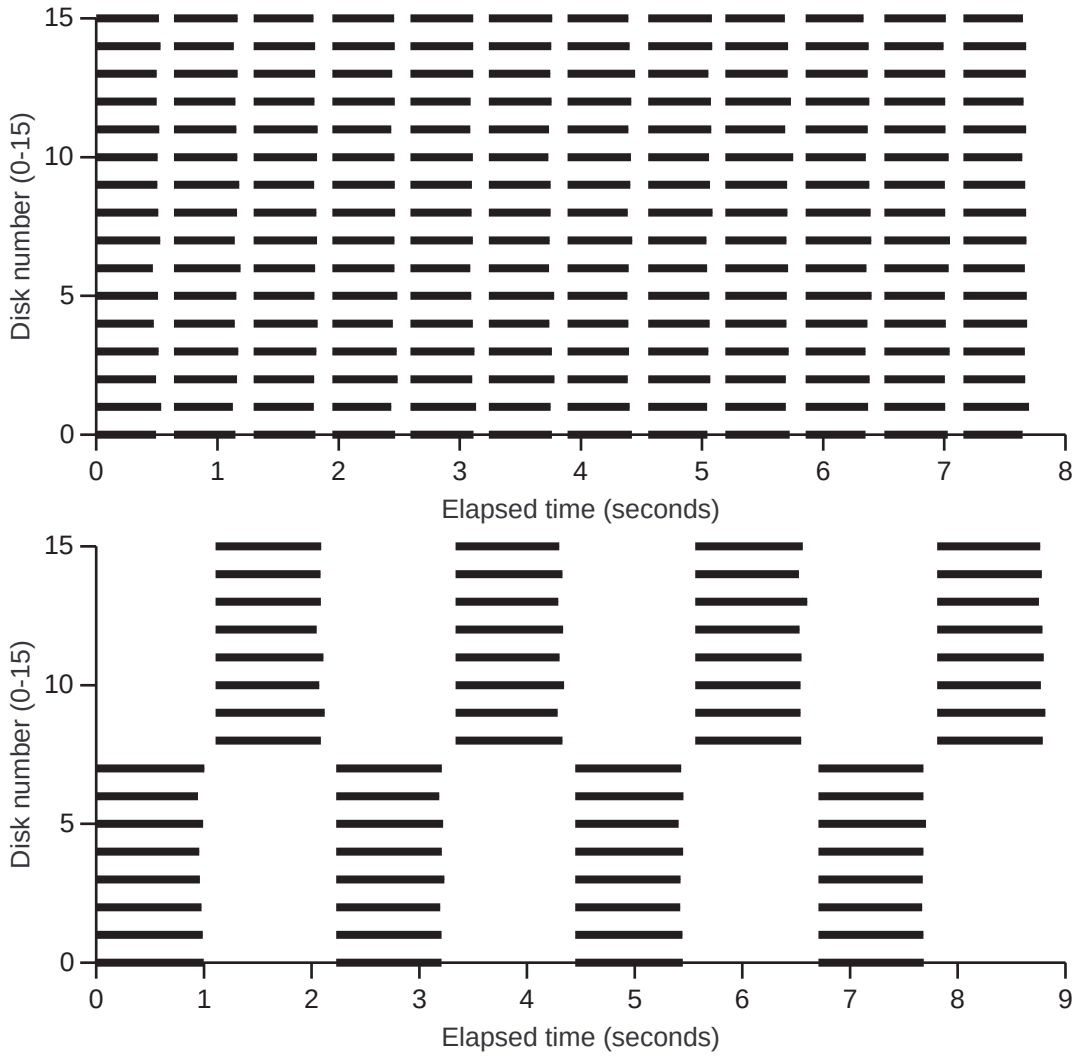

Figure 7-13. Temporal distribution of MPP file request streams to a striped file system.

Each dot in this diagram represents an access to the disk indicated on the vertical axis at the time shown on the horizontal axis. Both graphs show 16 processors reading a 1 GB file by accessing 1 MB per processor per iteration. The top graph shows the accesses resulting from iteration-sequential ordering, while the bottom graph shows node-sequential ordering. In both cases, all 16 disks are attached to a single node, with a stripe size of 2 MB. The system shown in the top graph achieves twice the bandwidth of the system in the bottom graph, as all disks are used for the majority of the time. In the bottom graph, however, half of the disks are idle at any time.

currently uses a system similar to Unix to manage the block allocation on each individual disk. The Intel Paragon file system [44] stripes data across many disks for better performance, but uses indices similar to those in FFS to get logical block numbers that are then mapped to physical disks. Thus, typical file systems, including standard striped file systems, need up to 20% of a disk free to ensure good performance.

In contrast, RAMA uses hashing to determine the disk line that each file block must go into. Since disk lines are smaller than an entire disk, the data is partitioned more than it would be in a striped file system. In the absence of tertiary storage, the file system would be considered full whenever any disk line filled up. Worse,

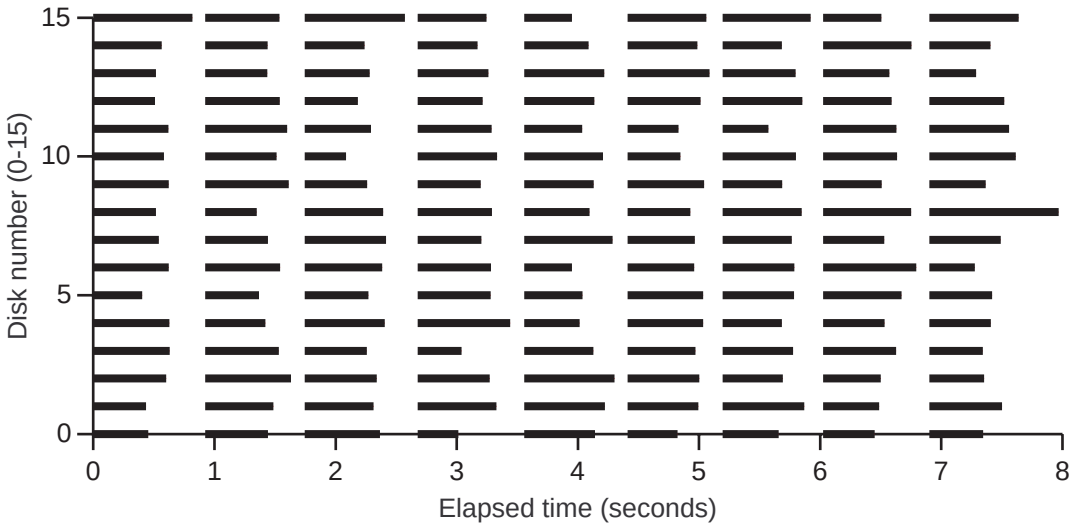

Figure 7-14. Effectiveness of the RAMA hash function at eliminating poor temporal distribution.

This figure shows the distribution of requests to disks for the same file access sequence shown in the lower graph of Figure 7-13. At any given instant, each disk has close to the average load for the entire disk system. As a result, no disk is idle for a long period of time.

it would be difficult for a user to predict which files had be deleted to free up space. Only files with blocks in the full disk line would alleviate the space crunch, yet a user would have no simple way of knowing which files those were.

To avoid these consequences, RAMA relies on tertiary storage to ensure that sufficient space is available for overwriting. Once a copy of a file exists on tertiary storage, the version on disk may be freed if space is needed. However, RAMA makes such data available so long as the space is not needed. RAMA was not simulated with a "tertiary disk;" however, analytical performance estimates for such a system indicate that the performance loss is no greater 1 MB/s per thrashing disk line. Additionally, this penalty is only paid if active data must be moved between the main RAMA disks and the tertiary disk. Otherwise, the penalty is paid once as inactive data is moved to the tertiary disk, and no further degradation occurs. So long as the active working set of data does not overflow many disk lines, using a "tertiary disk" has a minimal effect on RAMA's performance.

7.3. Network Utilization

Network utilization is also a major concern for parallel file system designers. Early file systems, such as the Concurrent File System [92] ran on MPPs with low interconnection network bandwidth. The nCube-2, for example, has links that run at 20 MB/s, only 5 times faster than the peak bandwidth of a disk. However, recent parallel processors such as the Intel Paragon [27,44] and the Cray T-3D [20] provide link bandwidths of 100 MB/s or more. Since peak disk bandwidths have not increased at the same rate, the interconnection network is no longer the limiting factor in file system bandwidth.

The load on a mesh network with the edges wrapping around can be approximated analytically under the assumption that the sending and receiving nodes are chosen randomly. If both are chosen randomly, an average network message will go one quarter of the total length in each direction. Thus, a network message on a 16 x 8 mesh will, on average, traverse 4 horizontal links and 2 vertical links, for a total of 6 links. The

expected total network bandwidth will be the product of the total file system bandwidth and the number of links each message must traverse.

7.3.1. Network Utilization Under Striped File Systems

Many MPP file systems in use today [5,20,52] attach disks to the interconnection network at a few dedicated I/O nodes. This approach concentrates link traffic due to the file system near those I/O nodes. For example, a file system that provides 200 MB/s through 8 I/O nodes must use 25 MB/s of network bandwidth near each I/O node. If the I/O nodes are part of the normal interconnection network, as they usually are, normal link traffic will be delayed in the vicinity of I/O nodes. Moreover, these delays decrease further away from the I/O nodes, as only file system data destined for nodes distant from the I/O nodes must use links far from the I/O nodes.

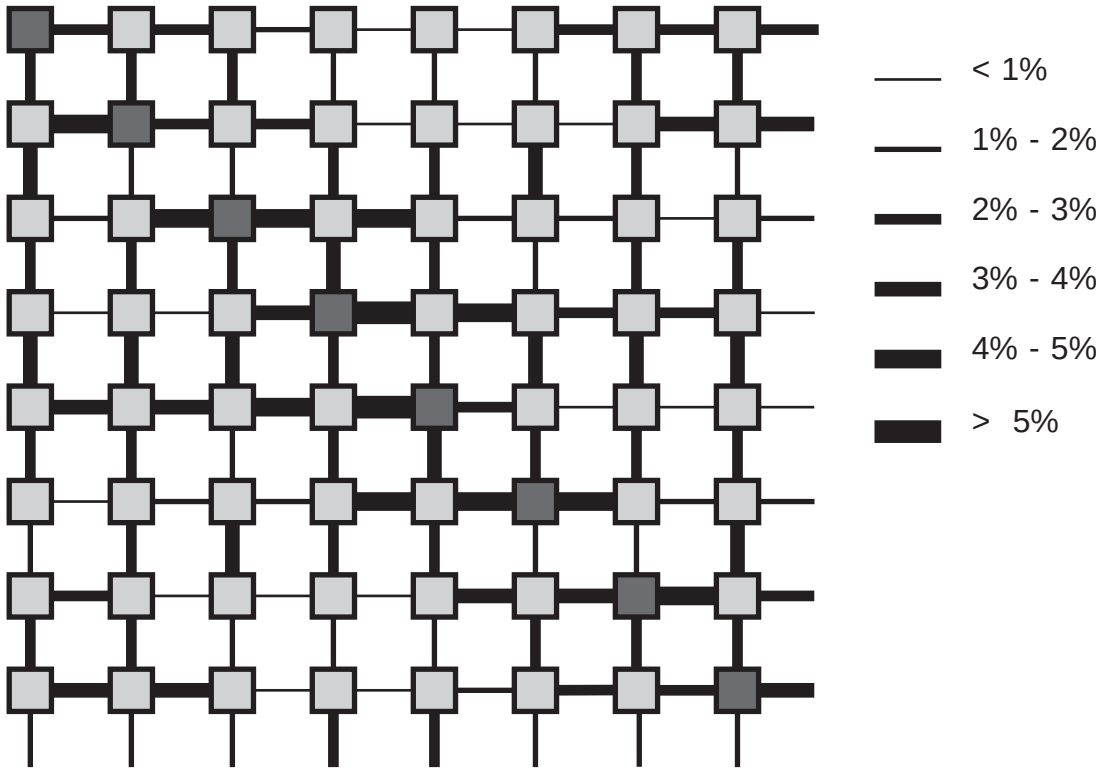

Figure 7-15. Interconnection network utilization under a striped file system.

This diagram shows the interconnection links for an 8 x 8 processor mesh with 8 I/O nodes running a striped file system. Each node that has I/O connections is darkly shaded, while the other nodes are lightly shaded. The thickness of a link between two nodes is proportional to the network traffic due to the file system carried by that link. The thickest links carry 5 MB/s or more.

The link weights are from a simulation of a 24,876 x 24,876 matrix decomposition. Network load is highest near the I/O nodes, and drops off further from the I/O nodes. The most heavily loaded link carried 5.5 MB/s on average, and the least-loaded node averaged 0.2 MB/s.

As Figure 7-15 shows, the file system's effect on the interconnection network varies according to a link's distance from the I/O nodes. Even though there are relatively few I/O nodes, the interconnection network must still carry all of the I/O traffic to the appropriate nodes. There are methods for restricting file system network traffic from each I/O node to its nearby neighbors, as mentioned in [68]. However, the application programmer must make extensive use of file system layout knowledge to make such improvements. Additionally, programs making these optimizations perform poorly if the disk configuration changes.

7.3.2. Network Utilization Under RAMA

The RAMA design, in contrast to striping, spreads the file system load on the interconnection network evenly around the MPP. Figure 7-16 graphically shows this balanced distribution. Since the file system data is evenly distributed among all of the nodes in the MPP, each node injects approximately the same amount of file system data into the interconnection network.

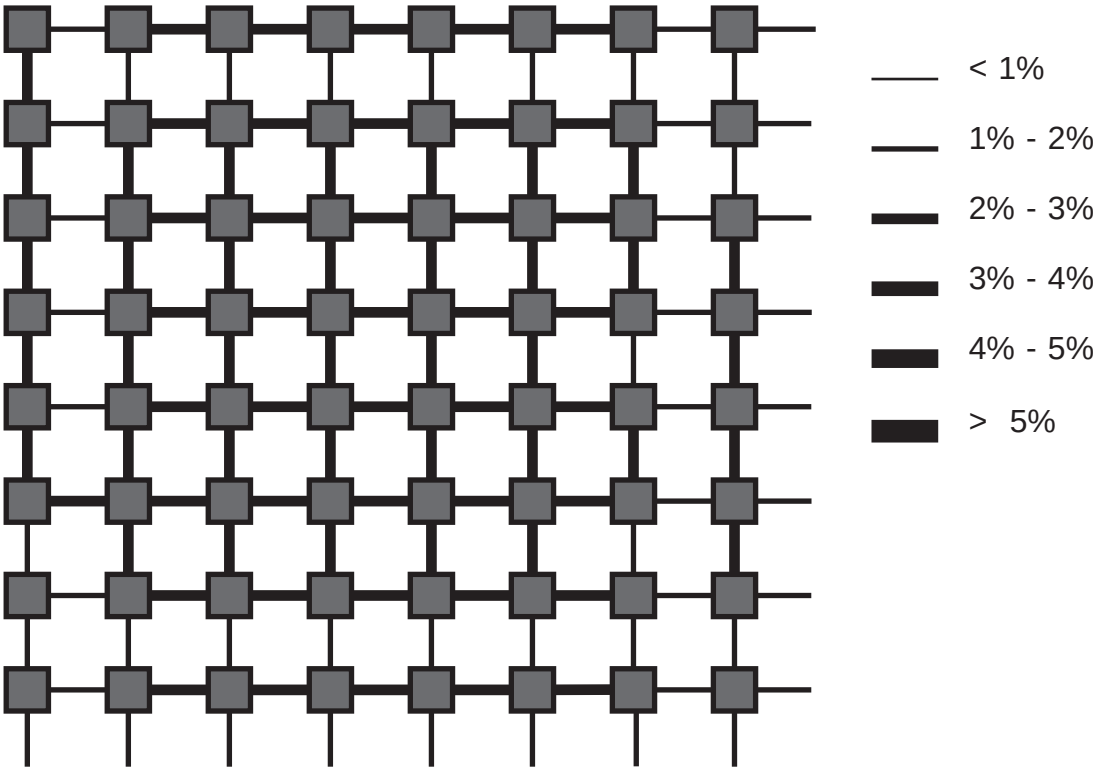

Figure 7-16. Interconnection network utilization under RAMA.

This figure uses the same conventions as Figure 7-15 for nodes and interconnection links. It shows the same algorithm — LU decomposition of a 24,576 x 24,576 matrix on an 8 x 8 processor mesh. However, the file system for this simulation is RAMA. Thus, every node has an I/O device attached to it. The network load injected by the file system is evenly distributed among all of the links. The most heavily loaded link carries 2.8 MB/s on average, while the least loaded link averages 1.6 MB/s.

RAMA provides an average of 69.2 MB/s of disk I/O for the program simulated in Figure 7-16. Since the mesh network it uses is 8 ×8, each network message should average 4 hops, for a total of 276.8 MB/s bandwidth distributed over the entire mesh network. The actual total network bandwidth is 277.8 MB/s. The estimate is only low by 0.4%. However, the bandwidth calculation ignores bandwidth due to messages

requesting data. While these messages are small, they do add about one hundred bytes per I/O request. If the average file request is for 32 KB, request messages should add 0.3% to network load. Once this is figured in, actual bandwidth varies by less than 0.1% from predicted bandwidth. The random source and destination model can thus be used to predict the file system load on larger mesh networks in MPPs running RAMA.

7.4. Conclusions

Traditional multiprocessor file systems use striping to provide good performance to massively parallel applications. However, their performance depends on the application to provide the file system with placement hints. In the absence of such hints, performance may vary by a factor of 4 or more, depending on the interaction between the program's data layout and the file system's striping.

RAMA avoids the problems of poorly configured striped file systems using pseudo-random distribution. Under this scheme, an application is unlikely to create hot spots on disk or in the network because the data is not stored in an orderly fashion. Laying files on disk pseudo-randomly costs, at most, 10-20% of overall performance when compared to applications that stripe data optimally. However, optimal data striping can be difficult to achieve. RAMA's performance varies little for different data layouts in full-speed file transfers and matrix decomposition. Applications using striped file systems, on the other hand, may increase their execution time by a factor of four if they choose a poor data layout. This choice need not be the fault of the programmer, as simply using a machine with its disks configured differently can cause an application's I/O to run much less efficiently.

RAMA's flexibility does not exact a high price in multiprocessor hardware, however. RAMA allows MPP designers to use inexpensive commodity disks and the high-speed interconnection network that most MPP applications require. It is designed to run on an MPP built from replicated units of processor-memory-disk, rather than the traditional processor-memory units. This method of building MPPs removes the need for a very high bandwidth link between an MPP and its disks; instead, the file system uses the high-speed network that already exists in a multiprocessor. Since the file system is disk-limited, though, the network is never heavily loaded. Network loads under RAMA vary by less than a factor of 2 across the entire MPP, while loads for traditional striped file systems vary by more than a factor of 10.

Disks, too, are utilized well in RAMA. Pseudo-random distribution insures an even distribution of data to disks. Disk requests are evenly distributed to disks in time as well as in space. Thus, no disk serves as a bottleneck by servicing too many requests at any time. In addition, all disks are used nearly equally at every step of an I/O-intensive application without the need for data placement hints.

The simulations of both synthetic traces and kernels of real applications run in this chapter show that the pseudo-random data distribution used in RAMA provides good performance while eliminating dependence on user configuration. While RAMA's performance may be up to 10-20% lower than an optimally configured striped file system, it provides a factor of 4 or more performance improvement over a striped file system with a poor layout. Its portability and scalability, however, make RAMA an excellent file system choice for the multiprocessors of the future.

8 Conclusions

This dissertation has addressed two issues in storage for scientific computing: the efficient management of terabytes of storage, and high-performance access to files in a massively parallel file system. It makes significant advances towards solving the challenges posed by the ever-increasing appetites of scientists for larger data stores and faster access to the data on them. This chapter will summarize the research described in this thesis and outline possible directions for future research in this area, focusing on problems that build on this work.

8.1. Summary

The first problem addressed in this dissertation was that of tertiary storage management for scientific computation. In the decade since the last studies of file migration and massive storage systems, both the volume of data and the rate at which it is produced and consumed have increased by orders of magnitude. The increased demands must be taken into account by new storage system designs. In addition, our analysis revealed other workload characteristics and performance metrics not found in previous analyses.

The traces for this study were gathered over a two-year period at the National Center for Atmospheric Research (NCAR). They include over three million references to 25 terabytes of data stored in nearly one million different files in the NCAR mass storage system. As in previous studies, about half of the files were referenced infrequently — two or fewer references. However, the remainder of the files experienced many rereferences. This behavior is similar to that reported around 1980 in [80].

Additional references to a file are most likely within a day or two of previous access to that file, though poor integration in the NCAR storage system forced two tertiary storage references regardless of how close together the two accesses occurred. There was little difference between the access patterns for large and small files. Reference patterns did differ, however, between reads and writes. File read rates closely matched scientists' schedules, peaking during the work day and dropping off at night and on weekends. On the other hand, file write rates remained relatively constant throughout the day and the week, suggesting that the write rate is governed more by the computer's ability to generate new data that must be stored.

We then shifted our attention to the design and simulation of a massively parallel file system motivated by the findings reported in Chapter 3. Current parallel file systems are not well-integrated with mass storage systems; however, Chapter 3 showed that more than half of the references to tertiary storage could have been avoided by transparent access to tertiary storage. Additionally, the storage of file metadata is becoming a problem. Many of the supercomputer file systems that allow transparent access to tertiary storage require all of the file system metadata and directories to remain on disk even when files are migrated to tape. The RAMA file system presented in this dissertation addresses both of these problems and adds another benefit: ease of use. In contrast to previous parallel file systems, RAMA requires little per-application customization to achieve high performance.

RAMA uses pseudo-random distribution and reverse indices similar to cache tags to uniformly spread data over the disks of an MPP. Because RAMA uses reverse indices to find file blocks, it only keeps metadata for files actually on disk; metadata for migrated files can itself be migrated. This strategy also allows seamless integration of tertiary storage, since a file can be retrieved from tertiary storage if it is not found in the disk-resident reverse indices. However, RAMA's major attraction is its performance. Its pseudo-random distribution probabilistically guarantees high bandwidth regardless of an application's access pattern. Unlike current MPP file systems, RAMA is designed to achieve high bandwidth without extensive placement and usage information from the applications that use it, thus providing an easy-to-use parallel file system.

After discussing RAMA's design, the dissertation described a simulation study of its performance. Chapter 5 discussed the methodology behind the simulation experiments performed in Chapters 6 and 7, covering both the simulator's organization and the models used for the disk and network devices. It also described the workload generators that generate reference streams to drive the simulator's model of RAMA. Several of the generators model real applications such as matrix decomposition, while others drive the simulation with synthetic access traces.

The next two chapters of the dissertation discussed RAMA's performance, showing that a pseudo-random file system can achieve bandwidth rates comparable to those delivered by conventional parallel file systems. Chapter 6 first discussed RAMA's sensitivity to changes in disk and network technology, demonstrating that it will remain a good file system choice even as the performance of the underlying technologies improves. As expected, faster disks resulted in proportionally higher bandwidths for RAMA. Faster networks, on the other hand, have little impact on RAMA's overall bandwidth because the file system is disk-limited. This finding is encouraging since interconnection link speed was a serious performance bottleneck in earlier parallel file systems.

In contrast to Chapter 6, which shows that RAMA will scale well with advancing technology and differing workloads, Chapter 7 demonstrated that pseudo-random distribution performs comparably to the striping layout used by previous parallel file systems. Moreover, striping performed well for some access patterns and poorly for others. RAMA, on the other hand, achieved bandwidth close to that of the optimal striping layout, and maintained that level of performance across a wide range of access patterns for which the optimal striping arrangement varied widely. RAMA's performance was generally within 10% of the best performance possible from striping, and was more than four times better than that of a file system with a poorly-chosen stripe size. It is this combination of high bandwidth and access pattern-independent performance that make RAMA an attractive alternative to conventional striped parallel file systems.

8.2. Future Work

The research presented in this thesis points to several areas for future research in storage for scientific computing. These directions include further research on algorithms and system design for tertiary storage systems and the construction and evaluation of disk-based storage systems for massively parallel computers.

8.2.1. Future Research in Tertiary Storage

Chapter 3 presented an analysis of the current situation at the National Center for Atmospheric Research. This data center provides just a single data point for tertiary storage systems. Since previous efforts in tertiary storage management have focused on optimizing the data movement around a specific storage hierarchy (different for each study), a clear direction for future research is to design file migration algorithms and test them using traces from several sites.

Initially, the file migration algorithms would only move whole files between different devices in the storage hierarchy. However, it is likely that allowing partial files to exist on various levels of the hierarchy will

result in better performance. This is particularly true when users are searching through many files for a specific piece of data because it allows the system to migrate only the necessary data from a multi-gigabyte file.

Since even a single "cache miss" for tertiary storage is visible to the user, another important area of research is prefetching algorithms. Again, the use of traces from several sites will allow the development of good heuristics that are based on general access patterns rather than the peculiarities of a single site. These algorithms will enable the mass storage system to hide from the user the multi-second delays incurred while fetching files from slow tertiary storage media.

The introduction of new storage technologies provides another direction for file migration research. How will the development of holographic storage and other high-speed, high-capacity inexpensive devices affect the design of storage systems? Certainly, computers will be able to store more data and retrieve it more quickly. However, the algorithms used in such a system may more closely resemble disk caching algorithms than those used in mass storage systems. Even before such devices are available, these questions can be considered in simulation.

8.2.2. Future Research on RAMA and Parallel File Systems

This dissertation showed the benefits of the RAMA file system design — pseudo-randomly distributing data throughout the entire parallel processor with at least one disk per node. Many questions remain, however.

First, these findings in this thesis are based on simulation. One major avenue for future research is the actual implementation of the RAMA file system on a parallel processor. This could be done on either a commercial MPP or a network of workstations tightly coupled by a high-speed network. As reported in Chapter 6, RAMA's performance is relatively insensitive to network latency; thus, the longer latencies of a workstation network would not result in much worse file system performance as long as network bandwidth was sufficient. Once the file system was completed, it could be used to verify the simulation results and demonstrate that RAMA could replace conventional parallel file systems.

Further experiments with parallel file systems demand additional traces. This dissertation used several common workloads; however, little is known about applications' use of parallel file systems. Thus, tracing and analysis of I/O-intensive parallel applications is another direction for future research. The results of such a study could then be used to drive additional experiments using RAMA.

RAMA itself also presents questions for additional study. Since the design described in this thesis is a paper design, it leaves several implementation details uncovered. Simulations can only model several hours (at best) of file system use; how will RAMA perform over the course of days and weeks? A major issue for long-term use is file system reliability, yet the RAMA design laid out in this paper is susceptible to disk failure. Techniques from RAID [11], the TickerTAIP project [9], and the Zebra file system [38] may be useful in adding reliability to RAMA, allowing data stored on it to survive disk failures.

8.3. Summary

This dissertation has addressed two issues in the design of mass storage systems for scientific computation: characterization of modern multi-terabyte storage systems, and high bandwidth file systems for massively parallel processors. These two areas are closely related, since both must be integrated into a storage environment for scientific computing. Building a coherent file system from many individual storage systems is as difficult a problem as is building each individual component. Thus, this thesis has also addressed issues of designing an integrated storage system for supercomputing. While performance and capacity are always recognized as important metrics of storage systems, ease of use is often neglected. Ideally, users should see a single file system entity, in contrast to current systems composed of several loosely-connected file systems between which files must be explicitly moved by users. Additionally, most current parallel file systems require a user to supply file layout directives for the file system to provide maximum performance. Future

massive storage systems supporting scientific computing must be easy to use as well as high-bandwidth and high-capacity.

The first third of the dissertation presents a detailed analysis of a modern tertiary storage system, providing a basis for the design and analysis of new algorithms for moving data between disks and tertiary storage devices such as tapes. Modern supercomputer centers store terabytes of data on tapes; however, their storage system designs and algorithms are *ad hoc*, since there is little research on modern systems on which to base their choices. This thesis has presented such an analysis, laying the groundwork for future research on file migration algorithms and mass storage systems designs.

Massively parallel processors (MPPs) have also created problems for designers of file systems for scientific computing. Recent advances in disk technology allow the storage of hundreds of megabytes on a disk smaller than a deck of playing cards, while modern high-speed MPP interconnection networks allow data to be stored further from its eventual destination with little performance penalty. The RAMA design proposed in this dissertation combines these two advances, yielding a parallel file system well-suited for scientific applications that provides high bandwidth without the need for complex user directives. RAMA is the first parallel file system designed to take *advantage* of tertiary storage rather than merely *support* it.

Bibliography

[1] Edward R. Arnold and Marc E. Nelson. “Automatic Unix backup in a mass-storage environment.” In *USENIX — Winter 1988*, pages 131–136, February 1988.

[2] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. “Measurements of a distributed file system.” In *Proceedings of the 13th Symposium on Operating System Principles*, pages 198–212, Pacific Grove, CA, October 1991.

[3] Amnon Barak, Bernard A. Galler, and Yaron Farber. “A hashed-index file system for a multicomputer with many disk nodes.” Technical Report 88-6, Dept. of Computer Science, Hebrew University of Jerusalem, May 1988.

[4] Kelly J. Beavers. “Helical scan recording technology: 8mm evolves.” *SunTech Journal*, pages 40–48, September/October 1990.

[5] Rajesh Bordawekar, Alok Choudhary, and Juan Miguel Del Rosario. “An experimental performance evaluation of Touchstone Delta Concurrent File System.” Technical Report SCCS-420, NPAC, Syracuse University, 1992. Also appeared in the 1993 International Conference on Supercomputing.

[6] Rajesh Bordawekar, Alok Choudhary, and Juan Miguel Del Rosario. “Design and evaluation of primitives for parallel I/O.” In *Proceedings of Supercomputing ’93*, pages 452–461, November 1993.

[7] Peter C. Boulay. “The LaserTape ’DOTS’: Next digital paper product.” Technical Report W1220, IDS Washington, Inc., 8000 Towers Crescent Drive, Suite 1180 / Vienna Virginia 22182, July 1990.

[8] Donald L. Boyd. “Implementing mass storage facilities in operating systems.” *Computer*, pages 40–45, February 1978.

[9] Pei Cao, Swee Boon Lim, Shivakumar Venkataraman, and John Wilkes. “The TickerTAIP parallel RAID architecture.” Technical Report HPL-93-25, HP Labs, April 1993.

[10] Peter M. Chen, Edward K. Lee, Ann L. Drapeau, Ken Lutz, Ethan L. Miller, Srinivasan Seshan, Ken Shirriff, David A. Patterson, and Randy H. Katz. “Performance and Design Evaluation of the RAID-II Storage Server.” *Journal of Distributed and Parallel Databases*, 2(3):243–260, July 1994.

[11] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. “RAID: High-performance, reliable secondary storage.” *ACM Computing Surveys*, June 1994.

[12] Ronald D. Christman, Danny P. Cook, and Christina W. Mercier. “Re-engineering the Los Alamos Common File System.” In *Digest of Papers*, pages 122–125. Tenth IEEE Symposium on Mass Storage Systems, May 1990.

[13] Sam Coleman and Steve Miller. “Mass storage system reference model: Version 4.” IEEE Technical Committee on Mass Storage Systems and Technology, May 1990.

[14] Bill Collins, Marjorie Devaney, and David Kitts. “Profiles in mass storage: A tale of two systems.” In *Digest of Papers*, pages 61–67. Ninth IEEE Symposium on Mass Storage Systems, November 1988.

[15] Bill Collins, Lynn Jones, Granville Chorn, Ronald Christman, Danny Cook, and Christina Mercier. “Los Alamos high-performance data system: early experiences.” Technical Report LA-UR 91-3590, Los Alamos National Laboratory, November 1991.

[16] J. P. Considine and J. J. Myers. “MARC: MVS archival storage and recovery program.” *IBM Systems Journal*, 16(4):378–397, 1977.

[17] George Copeland, William Alexander, Ellen Boughter, and Tom Keller. “Data placement in Bubba.” In *ACM SIGMOD Conference*, pages 99–108, June 1988.

[18] Peter F. Corbett, Sandra Johnson Baylor, and Dror G. Feitelson. “Overview of the Vesta parallel file system.” In *International Parallel Processing Symposium Workshop on I/O in Parallel Computer Systems*, pages 1–16, April 1993.

[19] Peter F. Corbett, Dror G. Feitelson, Jean-Pierre Prost, and Sandra Johnson Baylor. “Parallel access to files in the Vesta file system.” In *Proceedings of Supercomputing ’93*, pages 472–483, Portland, Oregon, November 1993.

[20] Cray Research, Inc. “Cray T3D system architecture overview manual,” September 1993. Publication number HR-04033.

[21] Cray Research, Inc. “Cray T3D: The right tool at the right time,” 1993. Publication number MCPB-1330893.

[22] Thomas W. Crockett. “File concepts for parallel I/O.” In *Proceedings of Supercomputing ’89*, pages 574–579, 1989.

[23] Juan Miguel del Rosario, Rajesh Bordawekar, and Alok Choudhary. “Improved parallel I/O via a two-phase run-time access strategy.” In *IPPS ’93 Workshop on Input/Output in Parallel Computer Systems*, pages 56–70, 1993. Also published in Computer Architecture News 21(5), December 1993, pages 31–38.

[24] Peter Dibble, Michael Scott, and Carla Ellis. “Bridge: A high-performance file system for parallel processors.” In *Proceedings of the Eighth International Conference on Distributed Computer Systems*, pages 154–161, June 1988.

[25] Peter C. Dibble. *A Parallel Interleaved File System*. PhD thesis, University of Rochester, March 1990.

[26] Ann L. Drapeau and Randy H. Katz. “Striped tape arrays.” In *Digest of Papers*. Twelfth IEEE Symposium on Mass Storage Systems, 1993.

[27] Rüdiger Esser and Renate Knecht. “Intel Paragon XP/S — architecture and software environment.” Technical Report KFA-ZAM-IB-9305, Central Institute for Applied Mathematics, Research Center Jülich, Germany, April 1993.

[28] Exabyte, Inc. “EXB-120 cartridge handling subsystem product specification,” 1990. Publication number 5120300-002.

[29] Robert J. Flynn and Haldun Hadimioglu. “A distributed hypercube file system.” In *Third Conference on Hypercube Concurrent Computers and Applications*, pages 1375–1381, 1988.

[30] Joy Foglesong, George Richmond, Loellyn Cassell, Carole Hogan, John Kordas, and Michael Nemanic. “The Livermore distributed storage system: Implementation and experiences.” In *Digest of Papers*, pages 18–25. Tenth IEEE Symposium on Mass Storage Systems, May 1990.

[31] Gordon George Free. “File migration in a UNIX environment.” Master’s thesis, University of Illinois at Urbana-Champaign, December 1984.

[32] Kathryn Fryberger and Marc Nelson. “Import and export of data on the NCAR mass storage system.” In *Digest of Papers*, pages 77–82. Ninth IEEE Symposium on Mass Storage Systems, October 1988.

[33] K. A. Gallivan, R. J. Plemmons, and A. H. Sameh. “Parallel algorithms for dense linear algebra computations.” *SIAM Review*, 32(1):54–138, March 1990.

[34] George A. Geist and Charles H. Romine. “LU factorization algorithms on distributed-memory multiprocessor architectures.” *SIAM Journal of Scientific and Statistical Computing*, 9(4):639–649, July 1988.

[35] General Atomics, DISCOS Division, P.O. Box 85608 San Diego, CA 92186. *The UniTree Virtual Disk System: An Overview*.

[36] W. Wayt Gibbs. “Ready or not: Holographic data storage goes to market - sort of.” *Scientific American*, 271(4):128–129, October 1994.

[37] Dirk Grunwald. “A users guide to AWESIME: An object oriented parallel programming and simulation system.” Technical Report CU-CS-552-91, University of Colorado at Boulder, November 1991.

[38] John H. Hartman. *The Zebra File System*. PhD thesis, University of California at Berkeley, 1994.

[39] Robert L. Henderson and Alan Poston. “MSS-II and RASH: A mainframe UNIX based mass storage system with a rapid access storage hierarchy file management system.” In *USENIX — Winter 1989*, pages 65–84, 1989.

[40] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1990.

[41] Alan R. Hevner. “Evaluation of optical disk systems for very large database applications.” In *Proceedings of SIGMETRICS*, pages 166–172, May 1985.

[42] Carole Hogan, Loellyn Cassell, Joy Foglesong, John Kordas, Michael Nemanic, and George Richmond. “The Livermore distributed storage system: Requirements and overview.” In *Digest of Papers*, pages 6–17. Tenth IEEE Symposium on Mass Storage Systems, May 1990.

[43] “IBM 9076 Scalable POWERparallel 1: General information.” IBM brochure GH26-7219-00, February 1993.

[44] “Paragon XP/S product overview.” Intel Corporation, 1991.

[45] David W. Jensen and Daniel A. Reed. “File archive activity in a supercomputer environment.” Technical Report UIUCDCS-R-91-1672, University of Illinois at Urbana-Champaign, April 1991.

[46] Randy H. Katz, Thomas E. Anderson, John K. Ousterhout, and David A. Patterson. “Robo-line storage: Low latency, high capacity storage systems over geographically distributed networks.” Technical Report UCB/CSD 91/651, University of California, Berkeley, September 1991.

[47] John T. Kohl, Carl Staelin, and Michael Stonebraker. “HighLight: using a log-structured file system for tertiary storage management.” In *USENIX — Winter 1993*, pages 435–448, January 1993.

[48] David D. Larson, James R. Young, Thomas J. Studebaker, and Cynthia L. Kraybill. “StorageTek 4400 automated cartridge system.” In *Digest of Papers*, pages 112–117. Eighth IEEE Symposium on Mass Storage Systems, November 1987.

[49] Duncan H. Lawrie, J. M. Randal, and Richard R. Barton. “Experiments with automatic file migration.” *Computer*, pages 45–55, July 1982.

[50] Edward K. Lee, Peter M. Chen, John H. Hartman, Ann L. Chervenak Drapeau, Ethan L. Miller, Randy H. Katz, Garth A. Gibson, and David A. Patterson. “RAID-II: A scalable storage architecture for high-bandwidth network file service.” Technical Report UCB/CSD 92/672, University of California at Berkeley, February 1992.

[51] Edward K. Lee and Randy H. Katz. “An analytic performance model of disk arrays.” In *Proceedings of SIGMETRICS*, pages 98–109, May 1993.

[52] Susan J. LoVerso, Marshall Isman, Andy Nanopoulos, William Nesheim, Ewan D. Milne, and Richard Wheeler. “*sfs*: A parallel file system for the CM-5.” In *Proceedings of the 1993 Summer Usenix Conference*, pages 291–305, 1993.

[53] Fred McClain. “DataTree and UniTree: Software for file and storage management.” In *Digest of Papers*, pages 126–128. Tenth IEEE Symposium on Mass Storage Systems, May 1990.

[54] Marshall K. McKusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry. “A fast file system for UNIX.” *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.

[55] Marshall K. McKusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry. *Unix System Manager’s Manual - 4.3 BSD Virtual VAX-11 Version*, chapter Fsck - The UNIX File System Check Program. USENIX, April 1986.

[56] Larry McVoy and Steve Kleiman. “Extent-like performance from a UNIX file system.” In *USENIX—Winter 1991*, January 1991.

[57] John Merrill and Erich Thanhardt. “Early experience with mass storage on a Unix-based supercomputer.” In *Digest of Papers*, pages 117–121. Tenth IEEE Symposium on Mass Storage Systems, May 1990.

[58] George Michael and Kenneth Wallgren. “Storage for supercomputers: A limiting issue.” *Optical Storage Technology and Applications*, 899:292–295, 1988.

[59] Ethan L. Miller and Randy H. Katz. “Analyzing the I/O behavior of supercomputing applications.” In *Digest of Papers*, pages 51–55. Eleventh IEEE Symposium on Mass Storage Systems, October 1991.

[60] Ethan L. Miller and Randy H. Katz. “Input/output behavior of supercomputing applications.” In *Proceedings of Supercomputing ’91*, pages 567–576, November 1991.

[61] Ethan L. Miller and Randy H. Katz. “An analysis of file migration in a Unix supercomputing environment.” In *USENIX—Winter 1993*, pages 421–434, January 1993.

[62] Eugene C. Nagel. “Digital storage technology: DAT evolves.” *SunTech Journal*, pages 52–60, September/October 1990.

[63] nCUBE. “nCUBE users handbook,” October 1987.

[64] nCUBE. “nCUBE 2 systems: Technical overview,” October 1992.

[65] Marc Nelson, David L. Kitts, John H. Merrill, and Gene Harano. “The NCAR mass storage system.” In *Digest of Papers*. Eighth IEEE Symposium on Mass Storage Systems, November 1987.

[66] John Ousterhout, Andrew Cherenson, Fred Douglis, Mike Nelson, and Brent Welch. “The Sprite network operating system.” *IEEE Computer*, 21(2):23–36, February 1988.

[67] John K. Ousterhout, Hervè Da Costa, David Harrison, John A. Kunze, Mike Kupfer, and James G. Thompson. “A trace-driven analysis of the UNIX 4.2 BSD file system.” In *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pages 15–24, Orcas Island, Washington, December 1985.

[68] Paul Pierce. “A concurrent file system for a highly parallel mass storage system.” In *Fourth Conference on Hypercube Concurrent Computers and Applications*, pages 155–160, 1989.

[69] James S. Plank. *Efficient Checkpointing on MIMD Architectures*. PhD thesis, Princeton University, June 1993.

[70] John S. Quarterman, Abraham Silberschatz, and James L. Peterson. “4.2BSD and 4.3BSD as examples of the UNIX system.” *Computing Surveys*, 17(4):379–418, December 1985.

[71] Steve Redfield and Jerry Willenbring. “Holostore technology for higher levels of memory hierarchy.” In *Digest of Papers*, pages 155–159. Eleventh IEEE Symposium on Mass Storage Systems, October 1991.

[72] J. Richards, T. Kummell, and D. G. Zarlengo. “A Unix-MVS based mass storage system for supercomputers.” In *Digest of Papers*, pages 108–113. Ninth IEEE Symposium on Mass Storage Systems, November 1988.

[73] Mendel Rosenblum. *The Design and Implementation of a Log-structured File System*. PhD thesis, University of California at Berkeley, 1992.

[74] Mendel Rosenblum and John K. Ousterhout. “The design and implementation of a log-structured file system.” In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 1–15, October 1991.

[75] Chris Ruemmler and John Wilkes. “An introduction to disk drive modeling.” *Computer*, 27(3):17–29, March 1994.

[76] A. Dain Samples. “Mache: No-loss trace compaction.” Technical Report UCB/CSD 88/446, University of California at Berkeley, September 1988.

[77] David S. Scott. “Out of core dense solvers on Intel parallel supercomputers.” In *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 484–487, 1993.

[78] Seagate Technology, Inc. *Spring 1993 Product Overview*, 1993.

[79] Margo Seltzer, Keith Bostic, Marshall McKusick, and Carl Staelin. “An implementation of a log-structured file system for UNIX.” In *USENIX—Winter 1993*, pages 307–326, January 1993.

[80] Alan Jay Smith. “Analysis of long term file reference patterns for application to file migration algorithms.” *IEEE Transactions on Software Engineering*, 7(4):403–417, July 1981.

[81] Alan Jay Smith. “Long term file migration: Development and evaluation of algorithms.” *Communications of the ACM*, 24(8):521–532, August 1981.

[82] Ken Spencer. “Terabyte optical tape recorder.” In *Digest of Papers*, pages 144–146. Ninth IEEE Symposium on Mass Storage Systems, November 1988.

[83] Stephen Strange. “Analysis of long-term UNIX file access patterns for application to automatic file migration strategies.” Technical Report UCB/CSD 92/700, University of California, Berkeley, August 1992.

[84] Eng Tan and Bert Vermeulen. “Digital audio tape for data storage.” *IEEE Spectrum*, October 1989.

[85] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.

[86] Erich Thanhardt and Gene Harano. “File migration in the NCAR mass storage system.” In *Digest of Papers*, pages 114–121. Ninth IEEE Symposium on Mass Storage Systems, November 1988.

[87] Thinking Machines, Inc. “Connection Machine model CM-2 technical summary.” Technical Report HA87-4, Thinking Machines Corporation, April 1987.

[88] Thinking Machines, Inc. *The Connection Machine CM-5 Technical Summary*. Thinking Machines Corporation, October 1991.

[89] David Tweten. “Hiding mass storage under UNIX: NASA’s MSS-II architecture.” In *Digest of Papers*, pages 140–145. Tenth IEEE Symposium on Mass Storage Systems, May 1990.

[90] Sandra J. Walker. “Cray Computer, MSS, MASnet, MIGS and UNIX, Xerox 4050, 4381 Front-End, Internet Remote Job Entry, Text and Graphics System, March 1991.” Technical report, National Center for Atmospheric Research, Scientific Computing Division, March 1991.

[91] David L. Williamson, Jeffrey T. Kiehl, V. Ramanathan, Robert E. Dickinson, and James J. Hack. “Description of NCAR Community Climate Model (CCM1).” Technical Report NCAR/TN-285+STR, National Center for Atmospheric Research, June 1987.

[92] Andrew Witkowski, Kumar Chandrakumar, and Greg Macchio. “Concurrent I/O system for the hypercube multiprocessor.” In *Third Conference on Hypercube Concurrent Computers and Applications*, pages 1398–1407, 1988.

[93] David Womble, David Greenberg, Stephen Wheat, and Rolf Riesen. “Beyond core: Making parallel computer I/O practical.” In *Proceedings of the 1993 DAGS/PC Symposium*, pages 56–63, Hanover, NH, June 1993. Dartmouth Institute for Advanced Graduate Studies.

[94] Tracy Wood. “D-1 through DAT.” In *Digest of Papers*. Ninth IEEE Symposium on Mass Storage Systems, October 1988.

[95] Ichiro Yamada, Minoru Saito, Akinori Watanabe, and Kiyoshi Itao. “Automated optical mass storage systems with 3-beam magneto-optical disk drives.” In *Digest of Papers*, pages 149–154. Eleventh IEEE Symposium on Mass Storage Systems, October 1991.