Issues and Problems in the I/O Subsystem
Part I - The Magnetic Disk

Robert Y. Hou, Gregory R. Ganger, Yale N. Patt
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor 48109-2122

Charles E. Gimarc
NCR Corporation, E&M - Columbia
West Columbia, South Carolina 29170

Abstract

It is well known that for many years, processor cycle times have continued to increase at a very rapid rate. On top of this, advances in multiprocessor technology have allowed potential system performance to increase at an even faster rate. The result, if we ignore the challenges in such fields as parallel algorithms, is that the performance of many of today's computer systems is limited by the I/O subsystem. In this paper, we attempt to do two things: (1) separate I/O space into three categories, based on their very different raisons d'etre and consequently very different characteristics, and (2) focus (the bulk of the paper) on the issues pertaining to improving the performance of one basic mechanism in the I/O subsystem, the magnetic disk. We do the former in order to set the framework of the I/O space. We believe that if we are to improve the performance of the I/O subsystem, we first have to understand the nature of I/O and not cloud our efforts by treating I/O as one homogeneous structure. We do the latter as the first step in dealing with the various mechanisms that make up the I/O space.

1 Introduction

A computing system is somewhat more than the sum of its individual pieces. Amdahl's Law argues that the performance of a computer system is driven by its weakest link. Historically, that weakest link could come from the algorithm specification, the language or compiler, the operating system, the instruction set architecture, the implementation of that instruction set architecture, including the processor, memory, and I/O subsystem, or the base technologies. Advances in recent years have put the blame, at least at the hardware level, on the I/O subsystem.

Microprocessor technology has improved from the Intel 4004 of two decades ago to today's Intel i860xp, a 50 MHz part that contains 2.5 million transistors on a single chip. This has allowed recent microprocessors to provide raw, unharnessed computing power comparable to state-of-the-art mainframes and supercomputers.

Concurrently, but not quite as dramatically, main memory densities and speeds have also improved [Myer86]. Main memory densities have increased at a rate of on the average 1.5 every year over the last 15 years. Main memory access times have decreased on the average between 1.3 and 1.8 during the same period [Asai86]. On the other hand, bus timing and physical topology have not improved to match. So, while memory technology has not quite kept pace with processor (i.e., cpu) technology over the years, it hasn't been the most serious sluggard in the computer system.

The I/O subsystem is a different kettle of fish. Some parts of the I/O system have done fine, such as the continuing advances in areal density on the surface of a disk. Harker [Hark81] has shown that improvements in areal density combined with increases in disk diameters have kept pace with increases in memory density. Bus technology, also, has provided improvements in the raw communication time available through fiber optics. Other parts of the I/O subsystem have not kept pace. For example, access time for disk drives, a function of seek time, rotational latency and transfer rate, has improved only minimally over the same time span [Hark81]. Data handling

capabilities of VDTs are already failing to support the data rates required by new applications. Networks, a boon to the utilization of spare computing power, introduce bottlenecks due to their protocols and data structures, necessary because of the mutually suspicious environments intrinsic to their nature.

Each of these bottlenecks is very different, and requires full treatment in its own right. In this paper, we address only the bottlenecks due to disk systems. We note that disk arrays have been in the computer system designer's toolbox for at least 15 years [Ouch78], and have been suggested over the years as the solution for many problems associated with storage. We review the hardware solutions proposed to support the efficient storage and retrieval of data. This includes discussion of both the basic mechanical characteristics of magnetic disk media and the architectural strategies that govern their use. In companion papers, we will address issues associated with networks and non-storage I/O.

This paper is organized in nine sections. Section 2 proposes a classification for operations outside the cpu/memory interaction that the I/O subsystem must support. We have noted above that the bottlenecks due to disk access time, VDT data handling, and network protocols are very different. Since storage devices, networks, and non-storage devices each have very different raisons d'etre and therefore very different characteristics, it is not surprising that the bottlenecks they create should also have very different characteristics. To understand these bottlenecks, so we can get about the business of removing them, we devote Section 2 to separating the I/O subsystem into these three major components: local storage (i.e., not involving a network), network activity, and non-storage. We hope to avoid clouding our efforts by not treating I/O as some homogeneous structure.

The next six sections all deal with storage, and particularly with the major mechanism for storage, the magnetic disk drive (and, in combination, the disk array). Section 3 briefly reviews the concepts and issues associated with a storage hierarchy. Section 4 narrows the discussion to the disk. Two important measures of disk subsystem performance and the importance of access time are discussed. Section 5 describes mechanisms for reducing the components of access time. Section 6 discusses some of the tradeoffs involved in designing disk array subsystems. Section 7 describes methods for avoiding accesses to disk drives. Section 8 reviews proposals for improving the reliability of I/O subsystems. Section 9 provides some concluding remarks.

2 A classification of I/O

We define the I/O space as all activity outside the interaction of a cpu and its memory. We partition I/O into three classes, as illustrated in Figure 1: storage I/O, non-storage I/O and network I/O.

Storage I/O can be characterized as operations that are

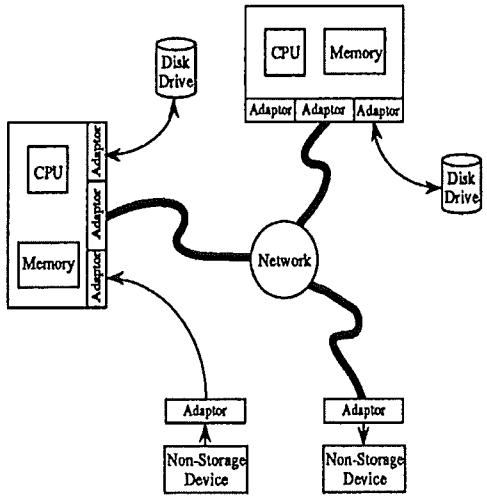

Figure 1: Three classes of I/O

performed on data that is stored for future use. The data element is bi-directional; it is written to the storage medium, and subsequently read from the storage medium. The ideal storage system can store an infinite amount of data and reliably retrieve it in a single cycle. Storage must be non-volatile so power is not needed to retain the data. Storage is intrinsic to the computer system. It acts deterministically on both input and output with the rest of the computer system.

Non-storage I/O can be characterized as operations that involve data that are always unidirectional, i.e., read-only or write-only. On input, non-storage I/O is non-deterministic. It is totally independent of the timing of the computer system. On output, non-storage I/O is produced for consumption, and not for future use. Devices that provide a communication link to the outside world perform non-storage I/O. Interaction with humans, with laboratory apparatus, and with process control applications are all examples of non-storage I/O.

Examples of the non-deterministic nature of non-storage input I/O are keystrokes on a keyboard and input data from a laboratory experiment. Keystrokes occur when the human operator wishes them to occur, totally independent of the timing of the computer system the keyboard is connected to. Input data from a laboratory experiment arrive at the time the experiment wishes, not according to the timing of the computer system.

Examples of the consumable nature of output I/O are data displayed on a CRT and data produced for use by a process controller. Once the data displayed on a CRT has been changed, the image on the CRT is not retrieved. In the case of a process control application, data produced by a computer system is consumed by the process controller,

and not retrieved at some later time by the computer system.

Network I/O can be characterized as operations that are performed to transmit or receive data from another computer system. Networks enable a computer system to utilize resources external to it. This permits computing power unutilized by the "owning" computer system to be utilized by some other "non-owning" computer system.

Because the two systems are distinct, the fundamental characteristic of network I/O is that the cooperating computer systems form a mutually suspicious hostile environment. Data that is transmitted from one computer system to another can consist of storage and non-storage I/O. Availability and the perceived response time of the data, however, is often beyond the control of the systems that are interacting, since the mechanisms to deal with this hostile environment add overhead to the time it takes to transmit and receive the data.

Examples of network I/O involve the sharing of resources. For example, multiple processors can cooperate on a single problem. Peripheral devices such as printers and modems can exist on one computer system but be used by multiple systems. A single software package can be physically stored on one system, but be accessible by any system connected to the network. Some fault tolerant schemes use networks as communication paths between components to prepare for eventual fault recovery when one computer system fails [Gray90].

Finally, it is possible for one application to utilize operations that involve the three distinct types of I/O when accessing data. For example, data stored on a disk on a remote system can be accessed via a network and displayed on a local CRT. The retrieval of data on the remote system requires storage I/O and network I/O, while displaying the data on the local CRT requires non-storage I/O. High performance execution of such an operation requires attention to potential bottlenecks in all three types of I/O.

3 Storage Hierarchy

We cannot build an ideal storage system. The best we can do is to give the illusion of having an ideal storage system by using a hierarchy of storage devices beginning with a semiconductor cache and main memory, including magnetic and optical disks, and ending with archival storage. The semiconductor cache and main memory provide the mechanism for quick retrieval of data. The disk and archival storage provide large capacity and non-volatility.

A storage hierarchy is generally characterized by cost and performance, where cost is expressed in terms of dollars per bit of data stored and performance is expressed in terms of time to access data from a storage device. Each level in this hierarchy provides a point in that space. Semiconductor cache, for those machines that contain a cache, has the highest cost and provides the highest level of performance in the storage hierarchy. Semiconductor main memory has a lower cost per bit and a slower access time compared to the cache. Magnetic tape has the lowest cost per bit and the slowest access time of the storage devices we discuss.

Where a data object resides within the hierarchy depends primarily on how often it is accessed and on its size. For example, a rarely accessed object should reside at the lowest level of the hierarchy. A small, regularly accessed object should reside at the highest level in the hierarchy. A larger, regularly accessed object could be stored at a high or low level depending on what other objects need to be stored and how often they are accessed.

Accesses to the storage subsystem, and especially to magnetic disks, incur a large time penalty which can result in degraded system performance. We can reduce this penalty by improving the access time of the disk, and by servicing I/O requests without accessing the disks themselves.

4 Disk Performance

4.1 Response time and throughput

The two most important performance metrics for storage I/O are response time and throughput. Response time is measured from request initiation to request completion. Throughput is the amount of data that can be transferred to and from the disk subsystem within a given period of time. Depending on the application, one measure may be considerably more important than the other. For example, in transaction processing, which is often characterized by numerous, small I/O requests, high throughput is the more important metric, allowing requests to be serviced independently and in parallel. For supercomputer applications, which are often characterized by a few, large I/O requests, response time is the more important measure. Response time and throughput both depend on the access time for disks, and there are many techniques for improving the access time.

4.2 Access time

When the computer system generates an I/O read request, it places the request in a service queue where it waits for the appropriate disk drive to become available. When the disk is available, a path is requested to send the read request from main memory to the disk. Depending on the I/O subsystem, this path can be comprised of several component paths, such as buses, each of which is responsible for passing the read request part of the way between main memory and disk. If the path consists of more than one component path, a request for a path to disk can be realized as several requests, each for a separate component path. The read request can be stored in a buffer if a request for a component path is pending.

When the read request arrives at the disk, the disk drive arm moves the read/write head to the target cylinder, waits for the target sector to rotate to its read/write head, and transfers the requested data to the disk drive buffer, if one is available, or directly to the path. The data is then transferred across each component of the path, although it may be stored in buffers if a component of the path is unavailable.

Much progress has been made in reducing the access time for a disk subsystem. The access time consists of the following:

- queuing time
- path time
- seek time
- rotational latency
- disk transfer time

Queuing time is the time a request spends waiting for the disk drive to become available. Path time consists of the time spent waiting for a path to or from a disk in addition to the time the path is used for data transfer. If the path consists of several component paths, the total path time is the summation of the path times for each component path. Seek time is the time required to move the disk read/write head to the target cylinder. Rotational latency is the time required for the requested sector to rotate to the read/write head. Disk transfer time is the time required to transfer data between the disk drive medium and the disk drive buffer or path. If an I/O request spans more than one track, multiple path times, seek times, rotational latencies and disk transfer times may be incurred. The next section reviews several techniques for reducing each component of the access time.

5 Reducing the Access Time

5.1 Reducing the queuing time

The queuing time is the time a request spends waiting for the disk drive to become available. The time depends on the other access time components and the workload. It can be reduced by providing more actuators to service I/O requests [Scra83]. Actuators can be added to existing disk drives. Alternately, disk drives can be added to the disk subsystem. Either method reduces the amount of data stored per actuator [Gray90]. It has often been observed that disk skew, the observed phenomenon where a few disks service most of the disk requests, may reduce the benefit of adding independent disks [Kim86]. Many researchers have suggested that interleaving data across multiple disks can more evenly distribute the requests (see Section 6 for more discussion).

Another method is to use shadowed disks [Bate85] [Bitt88]. Disk shadowing keeps N copies of the same data on N different disks. Each disk can then service a different I/O read request to the same data, thus reducing the queuing time for accesses to that data. The drawback to using shadowed disks appears when I/O write requests must be serviced, since all N disks must service each write request. As a result, the completion time for a write request to a set of shadowed disks is the completion time for the last disk drive to process that request.

5.2 Reducing the path time

The path time for an I/O request consists of two components - the time spent waiting for a path to become free, referred to as the path wait time, and the time spent using the path, referred to as the path hold time. If the path consists of several components, then the path wait time and path hold time are the sum of the path wait times and path hold times for the individual components. Path wait time is a function of the number of requests for the required path and the path hold time. It can be reduced by increasing the number of paths. If all else is constant, increasing the number of paths reduces path contention by permitting more than one simultaneous path request to be handled, increasing the performance of the disk subsystem [Ng88].

When a disk drive first initiates a seek to the target cylinder, it can disconnect from the controller, releasing the path to service other I/O requests. This reduces the path hold time, which in turn reduces the path wait time. After the disk arm has moved to the target cylinder, the connection is reestablished. The path is then held until the target sector rotates to the read/write head. Since rotational latency can be large, on the same order as seek time, many disk drives have hardware that allows the disk to disconnect from the controller while the target sector is rotating to the read/write head, again releasing the path to service other I/O requests [Ng91]. These disk drives have rotational position sensing (RPS) logic which allows them to detect when the requested data will be beneath the read/write head, at which time the disk requests reconnection. This also reduces the path hold time and path wait time.

If the controller or path is not available when the disk drive attempts a reconnection, an RPS miss occurs. The penalty for an RPS miss is the full rotation required before the target sector is again under the read/write head. RPS misses can be eliminated by placing a full-track buffer on the disk drive. For read requests, the data is read from the disk medium into the buffer as soon as the disk arm reaches the target cylinder, up to and including the target sector. Depending on the implementation, data following the target sector may also be read into the buffer. When the disk drive / controller connection is again established, data is transferred from the buffer to the controller. If the disk drive requests and establishes the connection to the controller before the data is completely transferred from the disk medium to the buffer, data transfer between medium

and buffer and data transfer between buffer and path can be overlapped. Overhead is incurred to transfer data from the medium to the buffer. By overlapping data transfers, this overhead is reduced.

Buffers provide another advantage since data can now be transferred at the speed of the buffer, which is usually substantially faster than the disk medium transfer rate. This can result in a significant reduction in path hold time, which in turn reduces path wait time. Another advantage is that buffers allow the disk drive to temporarily release the controller and path during a data transfer. This prevents a long data transfer from dominating the path, allowing other disk requests to be serviced. For example, the controller can initiate seek/rotate requests on idle disks [Cher90], reducing the path wait time.

Buffers also allow data to be transferred to a disk drive while the disk arm is moving to the target cylinder and the target sector is rotating to the read/write head. This reduces the perceived path time since the path time is overlapped with the seek time and rotational latency. This technique can only be applied for I/O writes.

Many current disk drives have buffers and hardware to implement the RPS feature, including Maxtor [Maxt89], Seagate [Seag90], and Fujitsu [Fuji90].

Other methods exist for reducing the path hold time, such as making the path faster and/or wider. Both changes will reduce the path hold time and path wait time. These are two improvements being made in the SCSI-2 bus protocol [NCR90].

5.3 Reducing the seek time

The seek time is the time required to move the disk drive read/write head to the target cylinder. The disk manufacturer specifies the average seek time required to move the read/write head from one track to any other track. Scranton et al. [Scra83], however, observed that the number of tracks a read/write head traverses when servicing I/O requests is often very small. In fact, for many requests, the head does not move at all.

There are several techniques for reducing the seek time. One method is to use shadowed disks [Bitt88]. Since there are N copies of the same data on N different disk drives, any of the N disks can provide data for an I/O read request. The disk drive whose actuator has the shortest seek time to the desired track is the one designated to service the I/O request.

When several I/O requests are queued for service, disk scheduling policies [Teor72, Geis87a, Bitt89] can be used to reorder them to minimize the average seek time, individual waiting time, and/or variance in waiting time. Several algorithms have been proposed, including First-Come-First-Served (FCFS), Shortest-Seek-Time-First (SSTF) and SCAN. FCFS is the simplest and fairest scheduling policy, but it often results in high average seek and waiting times. Shortest-Seek-Time-First, in which the request closest to the current cylinder is processed regardless of the direction the disk arm must move, has the lowest average seek time and can have the lowest waiting time. The drawback to SSTF is that requests can remain indefinitely in the queue if the load is high. SCAN is similar to SSTF except that the disk arm services the closest request in the queue as it moves across the disk in one direction. When it reaches the last track, it reverses direction and continues servicing requests. This policy is fairer than SSTF and attempts to reduce the variance in waiting times while still achieving low average seek and waiting times. SCAN is less efficient than SSTF for light workloads, since the disk arm moves back and forth from one end of the tracks to the other regardless of where the outstanding requests are. LOOK accounts for this by moving the disk arm in one direction until no more requests can be serviced in that direction, and then reversing direction. This improves performance over SCAN for light workloads. C-LOOK, a modification on LOOK, reduces the variation in the average waiting time by scanning the disk in one direction only. When there are no more requests to be serviced in that direction, the disk arm moves back across the disk without servicing any requests, and then begins again.

Geist et al. [Geis87] studied SSTF and SCAN, and suggested a modification of SSTF. A window of tracks can be defined starting at the current disk arm position. If there are requests lying within this window, the next request to be serviced is chosen from among these requests using the SCAN policy. If there is no request within the window, requests outside the window are considered using a variation on the SSTF policy where a penalty equal to the number of tracks in the window is paid if the disk arm must reverse its direction. For example, if there is a request on a track far beyond the window and another request nearby but in the opposite direction, the request in the opposite direction is serviced next.

5.4 Reducing the rotational latency

Rotational latency is the time taken for the desired sector to rotate to the read/write head. The average rotational latency is usually stated as 1/2 the time for one disk spindle rotation. The rotation speed has been roughly 3600 RPM for the last 20 years although state-of-the-art disk drives have reached 4400 RPM and even 5400 RPM [Seag91a, Fuji91]. Physical constraints such as higher power requirements and greater heat dissipation have made it difficult to increase this speed [Ng91]. In addition, more sophisticated electronics are required to support the increased data rate.

Ng [Ng91] has proposed several architectural strategies for reducing rotational latency. His first strategy uses mirrored disks, also known as dual copy or duplex disks, which are shadowed disks where the number of disk drives is two. If both disks are synchronized so their disk arms move together, their seek times are the same. In addition, if their spindles are synchronized so data on the two disks are 180 degrees out-of-phase, rotational latency for data can be re-

duced to 1/4 of a rotation instead of 1/2 as in the conventional case. The disadvantage to this approach, however, is that an I/O write request must be serviced by both disk drives. Ng proposes that non-volatile storage be added to the control unit to allow the second write to complete at a later time, independent of when the current I/O request is considered complete. If another I/O request is received by the disks, it is stalled until the second write completes.

A variation of this approach is to duplicate data within each disk drive. For example, data on one half of each track can be duplicated on the other half of the track. Again, rotational latency is reduced to 1/4 of a rotation. The disadvantage to this approach, as with the previous approach, is that disk storage is essentially halved. Non-volatile storage can be used to complete the second write as previously described.

One method for avoiding the storage loss associated with duplicated data is to put two diametrically opposing actuators on one disk drive. In this configuration, both heads move together to the same cylinder, with the head closest to the target sector selected to process the I/O request. An additional advantage is that an I/O write request only requires a single physical write since data is not duplicated.

5.5 Reducing the disk transfer time

The disk transfer time is the time required to copy data from the disk drive medium to a disk drive buffer, if one is available, or directly to a path. It is limited by the rate at which data passes under the read/write head. By increasing either the linear bit density of the disk drive medium and/or the rotational speed, the disk transfer time can be reduced.

Many researchers [Kim86, Livn87, Sale86] have suggested interleaving data, also known as striping data, across several disk drives. This allows parallel transfer of data to and from the disk medium, thereby reducing the disk transfer time. Large amounts of data can be quickly transferred to and from disks using this method. Kim [Kim86] examined the effects of interleaving data at the byte level across disks in which the actuators and spindles were synchronized. She compared this disk organization with a more conventional disk subsystem in which the disks were independent and disk requests were skewed and unevenly distributed across the disks. Kim made the interesting observation that a conventional disk subsystem becomes saturated before all the disk drives are fully utilized, due to the skewed nature of the I/O requests. An interleaved system, on other hand, is almost fully utilized when it reaches its saturation point, since all disks service all I/O requests.

Livny et al. [Livn87] examined the effects of interleaving data among disks at the track level, which they called declustering. They considered independent disks and investigated the effects of having simultaneous requests serviced by the disk subsystem. They also considered multiple-block requests.

Reddy et al. [Redd89] examined the effects of interleaving data at the byte level, which they termed disk synchronization, and interleaving data at the block level, which they termed declustering. Interleaving data at the byte level enables the disks to function together as a single logical disk when servicing a request. The seek time and rotational latency remain the same as for a single disk. Declustering data enables several disks to simultaneously service a multi-block request or enables them to independently service separate single-block requests. A set of disks can be separated into several groups. Disks within a group can cooperate to service individual requests, providing high transfer rates. Each group functions independently of the other groups and can service separate requests. Thus data is declustered between groups of disks and is interleaved within each group.

6 Some tradeoffs in disk array design

A set of disks can be treated as independent disks, a single large logical disk, or some combination of the two. Ng [Ng89] notes that the organization will affect the number of logical devices available to service I/O requests, which in turn affects queuing time. When the workload for a disk subsystem is light, the number of logical devices available is not a primary concern, and techniques can be used to reduce the access time. As an example, interleaving data at the byte level across two or more disks can be used to reduce the disk transfer time. This also distributes the load on the disks more evenly compared to a non-interleaved disk subsystem. In addition, the interleaved disks appear as one logical device. This is an advantage since it is easier to control one or a few logical devices than multiple independent disk units.

If, however, the disk subsystem workload is heavy and the I/O requests require small transfers, interleaving data at the byte level has disadvantages. Fewer logical devices are available to service I/O requests, negatively impacting the queuing delay [Ng89]. This occurs because the disk transfer time is a small component of the I/O access time, and requests benefit little from the decrease in transfer time. In addition, all interleaved disks must perform seeks and incur rotational latencies to process an I/O request, resulting in high disk utilizations and increasing the queuing time. Of course, when I/O requests involve large transfers, so the disk transfer time is a large component of the I/O access time, the reduction in access time provided by interleaved disks may compensate for the reduction in the number of logical devices and actually reduce the queuing delay.

If data is interleaved at the block level across a set of disks, I/O requests for data larger than a block will also require more than one disk drive to service them. Each disk will incur seek and latency overheads. The total overhead incurred for such I/O requests will be greater than what would be incurred by a single disk in a system consist-

ing of independent disk drives. This is one of the reasons Gray et al. [Gray90] proposes that data not be interleaved for systems requiring high throughput. If data is not interleaved, then only one disk drive is needed to service an I/O request, and only one disk drive incurs seek time and rotational latency. The total disk transfer time spent by the disk subsystem to service the request remains the same. Now, however, the other disk drives are free to service other I/O requests. In essence the interleaving or striping unit is infinitely large. Parity data is maintained for reliability. This parity data is interleaved across the disks for performance reasons (see Section 8 for further discussion). One caveat is that complexity is added to the software, which must appropriately place data on the disks to take advantage of this organization.

A similar tradeoff exists when using mirrored disks to reduce the rotational latency, as discussed in Section 5.4. Mirrored disks used in this way reduce the number of logical devices since both disks seek to the same target cylinder yet only one of them will actually service the I/O request. By reducing the number of logical devices, the system throughput may be reduced, especially under heavy workloads.

A different tradeoff occurs when spindle synchronization is considered [Kim91]. Assuming data is interleaved at the byte level across a set of disks and the disk arms on the separate disks move together to the same cylinder, rotational latency for the interleaved disks is the maximum of the latencies of all the disks. Ng [Ng89] determined that without spindle synchronization, the rotational latency for a disk array quickly grows with each additional disk and eventually becomes a full rotation. For small data transfers, rotational latency is often the dominant delay component, and therefore the effect of increased rotational latency is substantial. For large data transfers, where the rotational latency is not a dominant delay component, the effect is not as important and may even be insignificant.

7 Eliminating disk accesses

Perhaps the best way to improve the performance of disk accesses is to avoid them. A cache can effectively use the principles of temporal and spatial locality to provide fast access to disk blocks. Disk blocks can be cached at several places within the disk subsystem. Many operating systems, including UNIX, maintain a cache of recently used disk blocks in main memory. Ousterhout [Oust85] and Smith [Smit85] suggest that a large percentage of I/O requests can be serviced without going to disk. This reduction in the number of physical disk requests is determined by a number of factors, including cache size and write policy. For reliability, file data should always be written to disk immediately (a write-through policy). For maximal reduction of physical disk requests, file writes should be postponed for as long as possible (a delayed-write policy). UNIX compromises by periodically flushing dirty blocks to disk. Ousterhout [Oust85] found that by using a delayed-write policy, the number of I/O writes was reduced because file data is often overwritten or deleted shortly after being written. Alternately, systems that have I/O processors can keep a separate cache of disk blocks with the I/O processor [Miya86]. An advantage of this scheme is that main memory resources can be used for other purposes.

Disk blocks can also be cached on a storage controller such as the IBM 3880 [Gros85]. Drawbacks of this scheme are the software overhead to access the storage controller and path time between main memory and the storage controller. This overhead is incurred even for cache hits. Another potential drawback is that several paths can exist between main memory and the disk drives, such as in fault-tolerant systems. Since each path can pass through a different storage controller and have a different cache associated with it, cache consistency must be maintained [Smit85].

Another place to cache data is at the disk drive [Mits85, Smit85]. Many disk drives now contain this capability, including Maxtor [Maxt89], Seagate [Seag91], and Fujitsu [Fuji90]. In these cases, disk caches are small (4 - 8 tracks). They play a dual role of avoiding physical disk accesses as a cache and improving disk access performance as a buffer (as described in Section 5.2). These caches can be used to read the entire track containing the block to be read and to prefetch the next track. As with caching at the storage controller, cache hits incur software overhead and path time. Unlike storage controller caching, cache consistency is not a problem when data is cached at the disk drive.

There are some tradeoffs in deciding if disk blocks should be cached. Time required to search a cache and move disk blocks into the cache are two potential sources of overhead. Caching disk blocks is only beneficial when the reduction in physical disk accesses compensates for the overhead. If the hit ratio is poor, caching disk blocks can degrade overall performance [Smit85]. When the data being accessed exhibits poor locality, it may be appropriate to disable caching. It may also be appropriate to only cache data which exhibits the greatest amount of locality. Another tradeoff to consider is prefetching disk blocks, which can reduce performance. Requests for other disk blocks cannot be serviced during prefetching. Therefore, prefetching is only beneficial when there are enough requests for sequential disk blocks.

Another method for reducing the number of disk accesses is to combine I/O requests. This improves performance because seek time and rotational latency do not scale with request size. I/O requests that access sequential disk blocks can be combined as they are queued waiting for disk drives or for a path component in any part of the I/O subsystem. In fact, the file system can be designed to merge I/O requests. An example of this is the log-structured file system being developed at UC Berkeley [Oust89]. The log-structured file system treats the file space on disk as an append-only log. Conceptually, write

performance is greatly improved, because large numbers of writes are merged into a single disk access, and read performance is unaffected. This file system does have its own set of problems, however, such as garbage collection and the inability to use file placement strategies.

8 Reliability

Many techniques for improving the access time of an I/O request, such as those described in Section 5.5, involve the utilization of several disk drives. Reliability for an array of disk drives, however, is inversely proportional to the number of disks in the array [Patt88]. If a disk is damaged and needs to be replaced, a new disk must be loaded from tape. If data is interleaved at the byte level, no data blocks can be retrieved from the disk array until the damaged disk is replaced. If data is interleaved at a block level, data on the remaining disks may still be retrieved.

Several techniques have been suggested over the years for improving the reliability of disk drives. Recently these techniques have been labeled as RAID levels [Patt88, Katz89], where RAID stands for Redundant Arrays of Independent Disks. To provide disk arrays with high reliability using minimum redundancy, Ouchi [Ouch78] and Kim [Kim85] add a check-sum disk (not dissimilar to RAID level 3, which uses a parity disk). The check-sum disk is used to reconstruct the data on a damaged disk. RAID level 5 interleaves parity and data at the block level across disks to provide both high reliability and high performance. Interleaving data at the block level allows large requests to be serviced by all disk drives and potentially allows small I/O read requests to be simultaneously serviced by individual disk drives. Interleaving parity and data potentially allows multiple small I/O write requests to be simultaneously serviced. Lee et al. [Lee91] analyzed performance consequences due to parity placement.

If a parity disk is used to provide reliability for an interleaved disk array, and one of the disk drives is damaged, all disks must be accessed to reconstruct the data on the damaged disk and only one disk request can be serviced at a time. For applications, such as transaction processing, however, data must be reliably stored and quickly recoverable. Disk shadowing provides a highly reliable solution [Bate85] [Bitt88] since it keeps a copy of the data on more than one disk drive (RAID level 1). Thus any disk can satisfy an I/O read request while each disk must be updated for an I/O write request. If one disk becomes damaged, the other disks can immediately service future I/O requests. A new disk can be incorporated into a shadow set by copying one of the working disks.

The reliability of other components in a disk subsystem can also be improved through the use of redundancy. Pairs of controllers and duplicate paths connected to dual-ported disk drives provide a common configuration for achieving highly reliable disk subsystems [Bitt88]. Schulze [Schu89] shows that by organizing a disk array subsystem in a matrix with SCSI strings forming the columns, the reliability of a disk array can be further increased if the relationship between SCSI strings and parity is considered. For example, if parity is calculated using only one disk drive from each SCSI string, then the disk subsystem can recover if a SCSI string fails.

9 Conclusions

We have attempted to take a first step toward removing the bottlenecks in the I/O subsystems of modern computing systems. Our methodology is to provide a framework wherein the I/O subsystem can be treated in a systematic, coherent manner, and then to address each mechanism within that framework.

To do this, we first separate the I/O space into three components that we feel have characteristics that are inherently sufficiently dissimilar that treating them together would cloud our ability to understand the basic properties of the bottlenecks. They are storage I/O, network I/O, and non-storage I/O. Issues such as speed/bandwidth, latency and contention are of course important, but, we feel, only within the context of the larger classes described above.

Next, we begin the process of focusing on the issues pertaining to improving the performance of the various basic mechanisms in the I/O subsystem. Most of this paper is spent treating magnetic disks and disk arrays. We discuss both the basic mechanical characteristics of magnetic disk media and the architectural strategies that govern their use.

In companion papers, we will continue this process; i.e., we will address issues associated with other basic mechanisms within each of the three classes of I/O. We will also explore how the basic mechanisms cooperate on operations that transcend the I/O class boundaries.

Acknowledgements

This work was partially funded by NCR Corporation, E&M - Columbia. We gratefully acknowledge their support. This paper is one result of studies of I/O subsystems being performed jointly by researchers at NCR and the University of Michigan. The project originated from conversations with Jimmy Pike and Mark Campbell, and is being followed through by Roy Clark, all of NCR. Their interest and continuing support is greatly appreciated. We also wish to thank Joe Balazs, Ed Brown, Jim Browning, Dave Dyer, Mike Leonard, Bruce Worthington, Mary Lee Young and our research group at the University of Michigan for technical discussions on the various I/O issues.

References

[Asai86] S. Asai, "Semiconductor Memory Trends", *Proceedings of the IEEE*, December 1986, pp. 1623-1635.

[Bate85] K.H. Bates, M. TeGrotenhuis, "Shadowing Boosts System Reliability", *Computer Design*, April 1985, pp. 129-137.

[Bitt88] D. Bitton, J. Gray, "Disk Shadowing", *Proceedings of the Very Large Databases Conference*, September 1988, pp. 331-338.

[Bitt89] D. Bitton, "Arm Scheduling in Shadowed Disks", *COMPCON*, Spring 1989, pp. 132-136.

[Cher90] A.L. Chervenak, "Performance Measurements of the First RAID Prototype", U.C. Berkeley Technical Report UCB/CSD 90/475, May 1990.

[Fuji90] Fujitsu Limited, "M2622Sx/M2623Sx/M2624Sx Intelligent Disk Drives OEM Manual - Specifications & Installation", Specification Number 41FH5055E-01, December 1990.

[Fuji91] Fujitsu Limited, "M262X Product Brochure", January 1991.

[Fuji91a] Fujitsu Limited, "M2652 Product Brochure", 1991.

[Geis87] R. Geist, S. Daniel, "A Continuum of Disk Scheduling Algorithms", *ACM Transactions on Computer Systems*, February 1987, pp. 77-92.

[Geis87a] R. Geist, R. Reynolds, E. Pittard, "Disk Scheduling in System V", *Performance Evaluation Review*, May 1987, pp. 59-68.

[Gray90] J. Gray, B. Horst, M. Walker, "Parity Striping of Disk Arrays: Low-Cost Reliable Storage with Acceptable Throughput", *Proceedings of the 16th VLDB Conference*, August 1990, pp. 148-161.

[Gros85] C.P.Grossman, "Cache-DASD Storage Design for Improving System Performance", *IBM Systems Journal*, Volume 24, Numbers 3/4, 1985, pp. 316-334.

[Hark81] J.M. Harker, D.W. Brede, R.E. Pattison, G.R. Santana, L.G. Taft, "A Quarter Century of Disk File Innovation", *IBM Journal of Research and Development*, September 1981, pp. 677-689.

[Katz89] R.H. Katz, G.A. Gibson, D.A. Patterson, "Disk System Architectures for High Performance Computing", *Proceedings of the IEEE*, December 1989, pp. 1842-1858.

[Kim85] M. Y. Kim, A. M. Patel, "Error-Correcting Codes for Interleaved Disks with Minimal Redundancy", IBM Research Report, RC 11185, May 31, 1985.

[Kim86] M. Kim, "Synchronized Disk Interleaving", *IEEE Transactions on Computers*, November 1986, pp. 978-988.

[Kim91] M. Kim, "Asynchronous Disk Interleaving: Approximating Access Delays", *IEEE Transactions on Computers*, July 1991, pp. 801-810.

[Lee91] E. Lee, R. Katz, "Peformance Consequences of Parity Placement in Disk Arrays", *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 1991, pp. 190-199.

[Livn87] M. Livny, S. Khoshafian, H. Boral, "Multi-Disk Management Algorithms", *SIGMETRICS*, 1987, pp. 69-77.

[Maxt89] Maxtor Corporation, "XT-8000S Product Specification and OEM Technical Manual", Document 1015586, Revision B, July 1989.

[Mits85] A. Mitsuishi, T. Mizoguchi, T. Miyachi, "Performance Evaluation for Buffer-Contained Disk Units", *Systems and Computers in Japan*, Volume 16, Number 5, 1985, pp. 32-40.

[Miya86] T. Miyachi, A. Mitsuishi, T. Mizoguchi, "Performance Evaluation for Memory Subsystem of Hierarchical Disk-Cache", *Systems and Computers in Japan*, Volume 17, Number 7, 1986, pp. 86-94.

[Myer86] G.J. Myers, A.Y.C. Yu, D.L. House, "Microprocessor Technology Trends", *Proceedings of the IEEE*, December 1986, pp. 1605-1622.

[NCR90] NCR Corporation, "Understanding the Small Computer System Interface", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1990.

[Ng88] S. Ng, D. Lang, R. Selinger, "Trade-offs Between Devices and Paths In Achieving Disk Interleaving", *Proceedings of the 15th International Symposium on Computer Architecture*, 1988, pp. 196-201.

[Ng89] S. Ng, "Some Design Issues of Disk Arrays", *COMPCON*, Spring 1989, pp. 137-142.

[Ng91] S. Ng, "Improving Disk Performance Via Latency Reduction", *IEEE Transactions on Computers*, January 1991, pp. 22-30.

[Ouch78] N.K. Ouchi, "System for Recovering Data Stored in Failed Memory Unit", U.S. Patent #4,092,732, May 30, 1978.

[Oust85] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, J. Thompson, "A Trace-Driven Analysis of the UNIX 4.2 BSD File System", *10th Symposium on Operating System Principles*, 1985, pp. 15-24.

[Oust89] J. Ousterhout, F. Douglis, "Beating the I/O Bottleneck: A Case for Log-Structured File Systems", *ACM Operating Systems Review*, January 1989, pp. 11-28.

[Patt88] D. Patterson, G. Gibson, R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", *ACM SIGMOD*, May 1988, pp. 109-116.

[Redd89] A.L.N. Reddy, P. Banerjee, "An Evaluation of Multiple-Disk I/O Systems", *IEEE Transactions on Computers*, December 1989, pp. 1680-1690.

[Sale86] K. Salem, G. Garcia-Molina, "Disk Striping", *International Conference on Data Engineering*, 1986, pp. 336-342.

[Schu89] M. Schulze, G. Gibson, R. Katz, D. Patterson, "How Reliable is a RAID?", *COMPCON*, Spring 1989, pp. 118-123.

[Scra83] R. A. Scranton, D. A. Thompson, D. W. Hunter, "The Access Time Myth", IBM Research Report, RC 10197, September 21, 1983.

[Seag90] Seagate Technology, Inc., "Elite Disc Drive ST41600N User's Manual (SCSI Interface)", Seagate Technology, Inc., Publication Number 83327460-A, 1990.

[Seag91] Seagate Technology, Inc., "SCSI Interface Specification, Small Computer System Interface (SCSI) Elite Product Family", Seagate Technology, Inc., Document Number 64721702, Revision B, March 1991.

[Seag91a] Seagate Technology, Inc., "World Class Data Storage", Seagate Technology, Inc., Publication Number 1000-006, 1991.

[Smit85] A. Smith, "Disk Cache - Miss Ratio Analysis and Design Considerations", *ACM Transactions on Computer Systems*, August 1985, pp. 161-203.

[Teor72] T. Teorey, T. Pinkerton, "A Comparative Analysis of Disk Scheduling Policies", *Communications of the ACM*, March 1972, pp. 177-184.