DISK STRIPING †

Kenneth Salem
Hector Garcia-Molina

Department of Computer Science
Princeton University
Princeton, N.J. 08544

ABSTRACT

Just like parallel processing elements can substantially speed up computationally intensive tasks, concurrent transfer of data in and out of memory can speed up data intensive tasks. In this paper we study one general purpose facility for achieving parallel data motion: disk striping. A group of disks is striped if each data block is multiplexed across all the disks. Since each subblock is in a different device, input and output can proceed in parallel. With the help of an analytical model, we investigate the effect of striping on disk service times and its advantages and limitations in one of a set representative applications, file processing. We also explore several possible enhancements to striping: immediate reading, ordered blocks, and matched disks.

1. Introduction

A group of disk units is *striped* if each data block is stored, not on one of the disks, but across all of the disks. That is, if there are n disks, each block is split into n subblocks and a subblock is placed on each disk.

The goal of striping is to improve IO bandwidth. Ideally, if a block is spread across n disks, then it can be read (or written) in parallel, cutting access time by a factor of $1/n$. Unfortunately, this beneficial effect may be counterbalanced by several factors, including higher CPU overhead and larger rotational and arm movement delays. In this paper we study these tradeoffs in detail, and characterize the applications and circumstances that make striping advantageous. We also study variations and enhancements to the basic striping idea, and attempt to determine their usefulness.

In the current quest for very high performance computers, the idea of striping has been causing considerable interest. However, it is by no means a new idea. On some parallel-readout disk models, striping can be done, and has been for many years, within the disk drive itself. These units have multiple read/write heads mounted on a single arm, and parallel data transfers to and from the heads are possible. However, as disk densities increase, it becomes more difficult to align the different heads simultaneously during I/O. This makes parallel-readout disks much more expensive and limited to relatively small number of striping channels. The alternative, striping from separate disk drives, is becoming more effective, and is thus the only one we consider in this paper.

Striping (to different drives) has been supported by some operating systems. Although we have been unable to find references in the published literature, it appears that some early versions of Unix gave users the option of striping their files [9]. This option disappeared from the newer version of Unix, it seems, because it was not heavily used. More recently, the Cray Operating System (COS 1.13) supports striping [11]. Reportedly, it improves performance on some applications significantly. (As far as we know, this system was the first to use the term "striping" that we have adopted here.)

Striping has also been proposed for multiprocessors and database machines. For example, the PASM multi-processor [13] includes the specification of multiple disk drives for parallel loading of the primary memories. This scheme has the added advantage that each disk has an independent path to its corresponding memory module, avoiding the inherent contention of a system-wide memory bus. It is also interesting to note that Boral and DeWitt [2] point to striping as one important technique that may improve performance in database machines.

With rapidly increasing memory densities, there has recently been interest in memory resident database systems [6,7,8]. Here again, striping has been suggested as a means to improve bandwidth to disks. For example, striping can reduce the time it takes to load the database into memory after a crash, or it can reduce delays in writing the log.

Finally, we suspect that some type of striping is used in many high performance commercial applications, for instance, in sorting packages. However, in this paper we are interested in viewing striping as a *general purpose facility* that can be used in a transparent way. In this facility, applications would simply define the degree of striping (i.e., n) and would read and write full blocks. The manipulation of the subblocks would be invisible.

In spite of the wide range of applications of striping, very little is known about its performance and its usefulness as a system facility. Is it only useful for loading up databases and initializing large memories, or can it be used for other important computations? What variations of the basic striping idea work best? What are the limits in the performance improvements possible through the use of striped disks? Is striping simply a "hack", or should it be incorporated into operating systems? In order to answer these questions, we embarked on the present study. (Concurrently, M. Kim [10] has also studied striping, from a different but also interesting perspective.)

Most performance evaluations make numerous simplifying assumptions, but since we are dealing here with

† This work has been supported by NSF Grant ECS-8351616 and from grants from IBM and NCR corporations.

non-linear mechanical devices (i.e., the disks) we make more than our share. In general terms, we believe they are justified because we are only looking for general *trends* in striping performance. The present study must be viewed as an exploratory one, to be followed up by an experimental one that actually verifies the the promising ideas that we uncover here.

Due to space limitations, in this paper we are unable to present the full details of our model and all of our results. (These can be found in [12].) Instead we concentrate on the key features of the model and of striping, and present representative performance comparisons. Specifically, in Section 2 we outline our model for disks, including the CPU overhead, rotational and latency delays. Section 3 describes several striping enhancements that may improve performance. In Sections 4 and 5 we examine the "raw" service time for IO requests and the total time for the completion of a file processing task, varying the enhancements that are used. Finally, in Section 6 we make some concluding remarks.

In this paper we do not address the reliability aspects of striping. If no precautions are taken, then a single disk failure in a striped group makes the entire group inaccessible. However, error detection and correction codes can be easily added, reducing the magnitude of the problem. One possible strategy for failure recovery is briefly discussed in Section 6.

2. The Model of Disk Service Time

A simple mathematical model is used to determine the disk IO service time in a multiple-disk system. We define service time as the portion of the request response time during which useful work is being done. This does *not* include, for example, queueing times for requests or delays due to contention at the IO channel. The model attempts to be general in the sense that it can be set up to model different types of disk drives and different system configurations. Customizing the model to a given system is a simple matter of choosing an appropriate set of constants.

Service time consists of instruction execution time and the mechanical and electronic delays in the disk drive. In the remainder of this section we briefly discuss how each of these components is represented in the model.

2.1. CPU time

CPU time is the instruction overhead which is incurred with every disk IO operation. This includes such things as page location calculations and device driver execution. The amount of time spent executing such code varies widely from system to system and we do not attempt to model it in detail.

To reflect the CPU time, the model incorporates an initialization time $t_{ioc} + nt_i$. The first term represents that part of the CPU time that does not vary with the number of disks in the striped group. The second term represents processing which must be done for *each* disk in the n-disk system. For example, we may need to handle a "transfer completed" interrupt for each of the n disks.

2.2. Disk time

The remaining delays can be broken into three major components. *Seek time* is time required to position the read/write heads of the disk over the track containing the data. *Rotational latency* refers to the wait for the spinning platter to carry the desired portion of the disk track under the read/write heads. *Transfer time* is the time required to pull the data off of the disk and present it to the IO channel.

In modeling the seek and rotational delays, we assume that the size of the set of data blocks to which we need access remains constant as we vary the number of disks. The data is stored contiguously on each disk, and we refer to the space it occupies as the *task data space*.

Seek times and rotational latencies at each disk are treated as random variables. In determining their expected values, we assume that requested blocks are distributed uniformly through the task data space.† The model parameters relevant to this determination are presented in Table 2, together with the values that were used for most of the evaluations of Sections 4 and 5. Of these parameters, those directly concerned with the disk were assigned values corresponding roughly to actual values specified for DEC's RA81 disk drive [5].

symbol	description	value	units
t_{ms}	max seek time	50	msec
t_{os}	1-track seek time	7	msec
ω_{disk}	disk rot. speed	3600	rpm
h	# of disk heads	14	
c	# of cylinders	1258	
s	# of sectors/track	52	
d_s	bytes/sector	512	bytes/sector
t_{ioc}	CPU overhead	10	μsec
t_i	CPU overhead/disk	100	μsec

Table 2

The model used [12] captures the most important effects of striping on each of the three components. In general, seek and rotation delays for the same IO request will be different for each disk in a striped group.†† Since we must wait for all of the disks to respond to each request, expected delays for seek and rotation increase with the number of disks. On the positive side, transfer time will decrease since we retrieve less data from each disk. Another beneficial effect of increasing the number of disks is that the size of the task data space on each disk decreases. This leads to a decrease in expected seek times. In section 4 we examine the importance of each of these effects on the overall performance of striped disks.

3. Performance Enhancements

Early data from the model confirmed that seek delays and rotational latency were the major contributors towards the response times of striped disk groups. With an eye towards reducing these times, several enhancements to the original striped disks were developed and the model expanded to include them. Each of the enhancements acts to reduce either the seek delays or rotational latency of the disks. They can be included in the model in any combination, so that the effect of each on system performance can be determined. These enhancements are discussed in the

† This assumption is modified when we consider ordered blocks, described in section 3.

†† In the next section, we consider a scenario in which this is only partially true.

following sections.

Note that some enhancements may be easier to implement than others. We return to these implementation questions after Section 5, once we understand the performance gains that can be achieved with each enhancement.

3.1. Immediate Reading

The original model assumes that any transfer of a block of data to or from a disk must begin at the position of the beginning of the block on the disk. Therefore, if the disk head arrives at the proper track just after the beginning of the desired block has spun past, data transfer is delayed until the disk has made almost a complete revolution. If instead we assume that the capability exists to begin data transfer from any point within the block, we can reduce the expected rotational latency of the disk. We call this the *immediate reading* enhancement.

With immediate reading the expected rotational latency varies with the ratio of subblock size to disk track size. In fact, if we match the block size to the track size, we reduce rotational latency to zero since we never have to wait for the correct part of the track to spin under the read/write head.

Note that immediate reading could also be used to enhance the performance of a single disk. In a striped group, however, rotational delays are more critical, since we must always wait for the slowest of the n disks in the group. Therefore we expect that enhancing a striped disk group in this manner will produce a more dramatic speedup than would the enhancement of a single unit.

3.2. Ordered Blocks

In this scenario, data is stored on the disk in the order in which it is to be retrieved. Frequently, there is some natural order to the data in an application. For example, a b-tree can be stored on disk one level at a time so that the disk head need never "backtrack" over previously visited cylinders to get the next block. Sometimes it doesn't matter in what order the data is read, for example if we are trying to compute the average salary found in a database of employee records. In these cases we can merely specify that the data is to be read in physically sequential order.

Typically, the price that is paid for this speedup is that adding and deleting data becomes more difficult. Going back to the b-tree example, we must either include extra blocks to handle "spill-over" data or rearrange the b-tree on the disk (presumably an expensive operation) when additions and deletions occur. However, ordered blocks may still be useful if the percentage of operations that cause modifications is small [14].

An important consequence of ordering disk blocks is that seek times can no longer be considered probabilistic in the same sense as before. With ordered blocks there is more information as to the location of the next block to be read. In some cases the location may be completely specified.

Ordered blocks can also be used to enhance the performance of single disks. However, as was the case for the immediate reading enhancement, we expect that a striped group employing ordered blocks will show a more dramatic performance improvement.

3.3. Matched Disks

Here we consider the possibility of requiring that the pieces of a block be stored at the same location on each of the disks. This way, the read/write heads of all of the disks will move together. With matched disks the expected seek time will remain approximately constant as the number of disks in the system increases, since all of the disks are doing the same seek.

4. Results - Service Time

The effectiveness of striping is the percentage of speedup obtainable by striping data across n disks rather than storing it on a single disk. This is measured by comparing the service time in the striped system to the service time in the other. For example, consider a situation in which the service time for a single-disk system is 25 milliseconds, while the service time is 18 milliseconds for a system with a 3-disk striped group. We compute the effectiveness of the striped group by:

$$effectiveness = \left\lfloor \frac{25 - 18}{25} \right\rfloor 100 = 28\%$$

Under this definition, the maximum possible speedup is 100%. Negative effectiveness indicates that the task completion time for the striped group is greater than that for the single-disk system.

Figure 4a shows the effectiveness of striping for block sizes ranging from 1K to 256K bytes. Unless otherwise noted, the parameters of Table 2 are used. It is clear that striping is more effective for larger block sizes, although significant savings can be obtained for block sizes as small as 1K bytes. That large block sizes provide an ideal environment for striping is to be expected since transfer time is exactly the component of the task completion time that striping works to reduce.

Figure 4b shows the break down of service time into its component parts. The figure is for a block size of 16K bytes. It is clear from this figure that most of the savings result from the decrease in transfer time as n increases. It is also interesting to note that the seek and rotation time is actually decreasing as the number of disks gets larger. In this case the effect of the decreasing size of the data space outweighs the delays expected from the wait for the slowest of the disks.

In Figure 4c we show the effect of the various enhancements on striping's effectiveness. Here we use a block size of 4K bytes. We see that the immediate reading enhancement actually results in a slight decrease in effectiveness in this situation. This somewhat unexpected result arises because of the shrinking size of the data subblocks as n increases. As was noted previously, the rotational latency of a disk with immediate reading capability is a function of the ratio of the subblock size to the track size. We caution that this result does not imply that service time is longer with striped disks, only that striping is slightly less *effective* with immediate reading than without.

4.1. Other Disk Drives

This section is included to show how varying our choice of disk drives affects striping's performance. Two additional drives, the RA80 and the RP06, are compared to the RA81 drive, which was used in the previous comparisons [3,4,5]. Values of parameters for all three drives are listed in Table 4 for comparison.

Figure 4d shows striping effectiveness when using the various disks, for a block size of 32K bytes. An

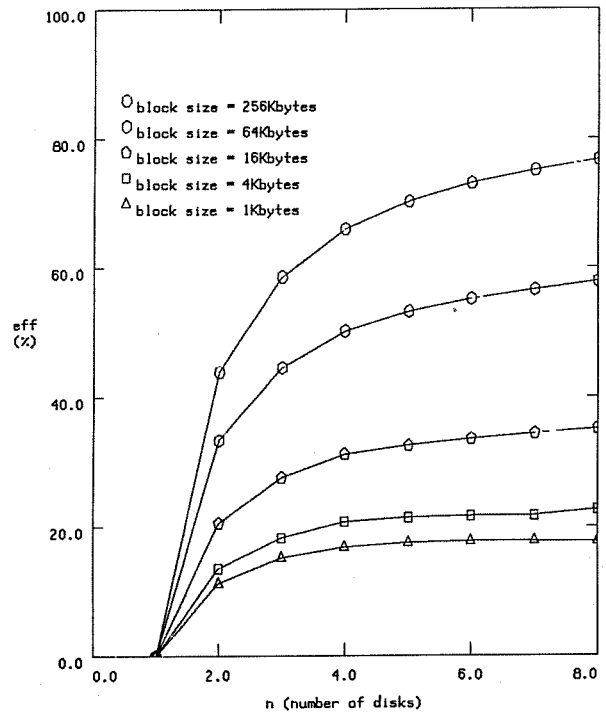

Figure 4a

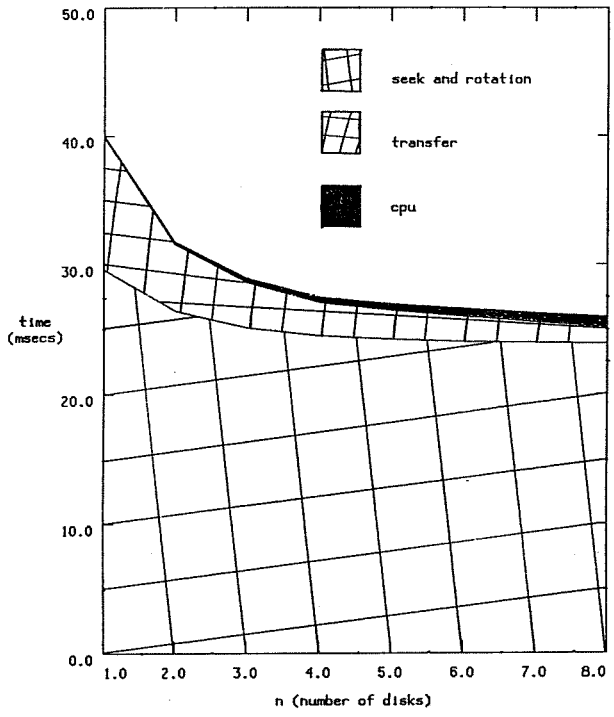

Figure 4b

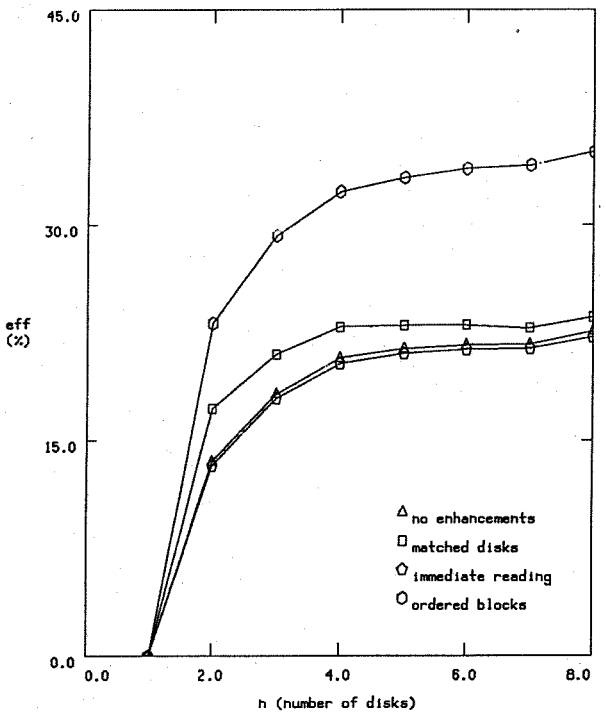

Figure 4c

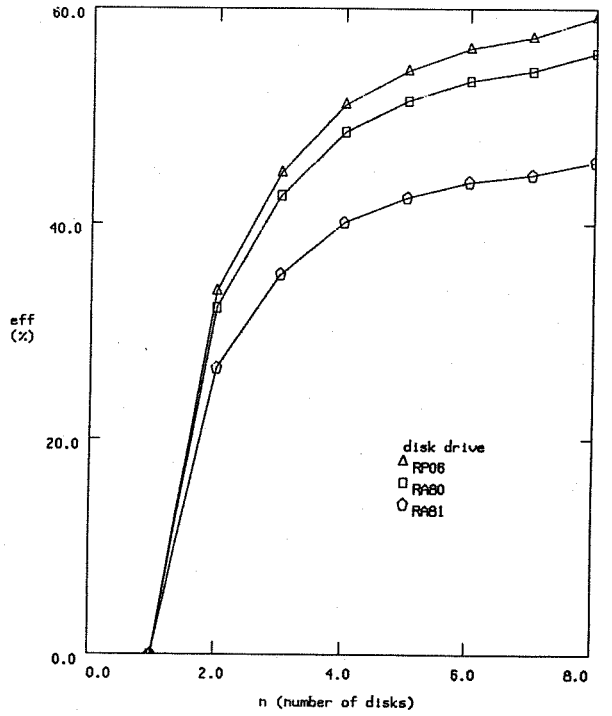

Figure 4d

parameter	RA81	RA80	RP06	units
t_{ms}	50	50	60	msec
t_{os}	7	6	10	msec
ω_{disk}	3600	3600	3600	rpm
h	14	14	19	
c	1258	561	815	
s	52	31	22	
d_s	512	512	512	bytes/sector

Table 4

examination of the characteristics of the disk drives reveals why some are more advantageous to striping than the others. Simply stated, if their rotational speeds are constant then striping is more effective with drives having lower bit-per-track densities. This is so because, for a fixed block size a lower bit-per-track density means a longer transfer time. This is the situation for which striping is the most effective, since it acts to reduce transfer time.

The same statement could also be made for disks with slower rotational velocities, since this again would translate to longer transfer times. However, the effect is partially counterbalanced in this case because slow rotational velocity also means longer expected rotational latency. This leads to less effectiveness since striping tends to bring out the worst in rotational delays.

5. Results - Applications

To fully evaluate striping we must look at an entire application, not just at an isolated disk to memory data transfer. Therefore we selected several tasks representing a variety of data access and computational patterns. Each was analyzed using the model presented in the last section to determine the effectiveness of striping in that particular situation. Most of these tasks involved multiple disk IO operations. Each included some measure of processing time once the data was present in memory.

Effectiveness for an application is computed the same way it was computed for service times, except that now we are comparing the total times to complete the task in the striped and unstriped systems

Due to space limitations, we only describe one of these tasks here, which we call file processing, and present some of our results. (The conclusions of the following section, however, are based on studying other tasks too.) The file processing task is to read and perform some type of operation on the records in a file. Depending on which of two variations of the task was chosen, the processor then may or may not be required to write the records back out to the disks. For all of the data presented in this section the latter variation is to be assumed unless otherwise specified

The file processing task requires several unique input parameters. These are listed in Table 5.

symbol	description	value	units
r_d	database size	10^8	bytes
m	buffer size	32K	bytes
p	CPU time/byte	1	μsec

Table 5

Processing of the data is simulated by the specification of a processing constant p, the amount of time necessary for the processing of each byte. The number of bytes of memory available for the task is specified by the parameter m. The block size is not specified as a parameter. Instead, for each n computations are done for a variety of block sizes and the smallest time is used as the completion time for the task on an n-disk system. Block size is restricted to be no more than one-half of the available memory buffer space so that we have room for double buffering.

It is assumed for this task that disk requests can be queued. The processor initially requests enough blocks to fill the memory buffer, then begins processing the first block. After each block is processed, a request to write the used block back to disk is queued, followed by a request to read in a new block if any remain. Processing of the next block is then begun. Letting k represent the number of blocks that can fit in the memory buffer, we summarize this protocol with the simple program shown in Program 5.

```
initial phase:
    REPEAT k TIMES {
        IF (unrequested blocks remain) THEN
            queue a block read request
    }
main phase:
    WHILE (not all records processed) DO {
        process a block in the buffer
        queue a block write request for that block
            [optional]
        IF (unrequested blocks remain) THEN
            queue a block read request
    }
```

Program 5

No scheduling optimization is modeled for the disk request queue; requests are assumed to be handled on a first-come-first-serve basis. An estimation as to whether the completion time will be IO-bound or processor-bound is used in determining the completion time.

Some results from the file processing model are shown in Figures 5a-b (again using the parameters of Table 2). Like most of the effectiveness curves in this paper, those for the file processing task tend to flatten out as n increases. Two factors contribute to this tendency. First, the expected value of the sum of seek and rotation time in a striped system approaches a fixed maximum value, namely $t_{ms} + 1/\omega_{disk}$, as n increases. Second, as the number of disks increases, the subblocks become so small that transfer time becomes negligible. Where these effects become noticeable depends on the characteristics of the system. After that point, the addition of additional disks to the striped group causes little change in its effectiveness.

The first figure shows the important and somewhat unexpected effect of memory size on effectiveness. When only an 8K buffer is available, blocks are limited to only 4K bytes and striping is not worthwhile. With 32K available, striping becomes effective. However, if memory is increased beyond this value, effectiveness decreases once again! This is because the extra memory lets us decrease IO times to the point where IO is no longer a bottleneck. This is an important property of

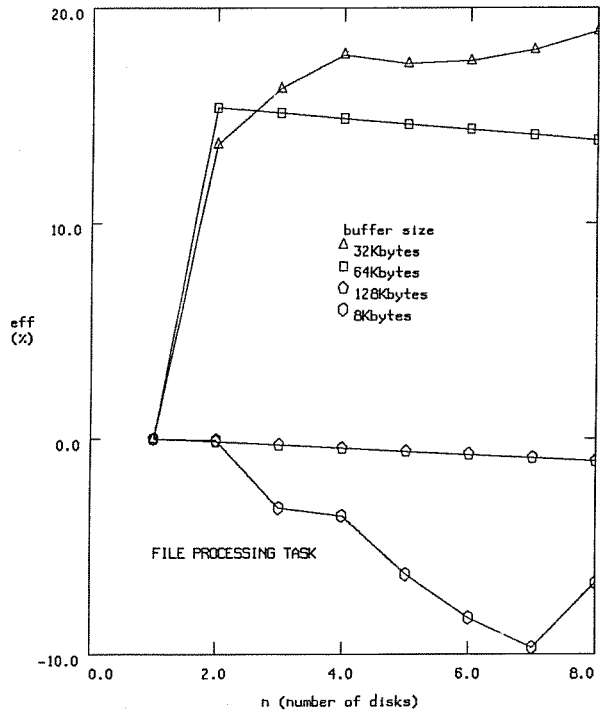

Figure 5a

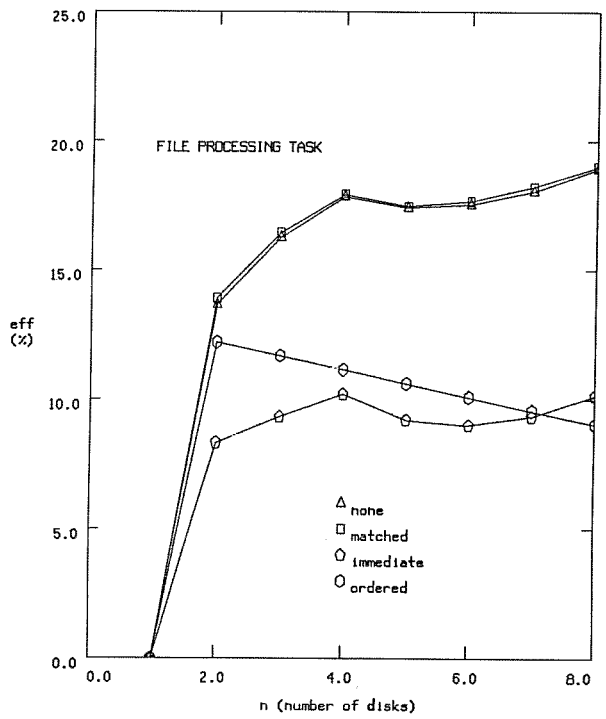

Figure 5b

any system in which IO and processing are occurring simultaneously: once the task is no longer IO-bound, adding additional disks to the striped group will not change its effectiveness. With 64K bytes of memory, the disks begin to outrace the CPU when $n = 3$. With 128K, even when $n = 1$, the CPU cannot process the data as quickly as the disk can supply it.

Figure 5b shows striping's effectiveness under the various enhancements, using a buffer size of 32 Kbytes. We can see that the curve for ordered blocks behaves much as did the curves for large buffer sizes in the previous graph. Again, we have decreased the IO time to the point where it is no longer the bottleneck for the completion time. Because seek times are relatively small in comparison to other factors (e.g. rotational latency) in this example, we find that matching disks does not significantly change effectiveness, and the negative effect of the immediate reading enhancement is more pronounced. With larger file sizes or smaller disks, both enhancements would allow greater effectiveness.

6. Conclusions

In addition to the results presented here, we have studied striping in considerable detail, varying the model parameters, studying other striping enhancements, and analyzing two additional tasks: B-tree searching (requiring relatively little processing) and merge-sort (requiring substantial processing) [12].

In general terms, our results on striping seem to be mixed. On the one hand, the performance improvements we have observed for the "typical" tasks are in 10% to 70% range (speedups of up to about 3 to 1), modest in comparison to the orders of magnitude improvements sought by supercomputer efforts. On the other hand, the striping improvements have been achieved at a very low cost. Many existing computers already have multiple disks and controllers, so the hardware cost of striping may be zero. The software development cost may also be small, especially if a general purpose striping facility is implemented and its cost amortized over many applications.

Striping also promises to become much more effective in the near future. Processors are becoming faster, memories larger, but disk speeds are not changing very much. At the same time, data requirements are growing in many applications. All of these factors make striping more effective. For example, in Figure 6 we plot the effectiveness of striping not for a typical file processing task as was done in Section 5, but for a task that is likely to be encountered in 5 years. We have set the processor speed at one nanosecond per byte (achievable with parallel processing), the file size at 2 gigabytes ,and the memory buffer at half a megabyte. (The curve begins at $n = 5$ since we need five of our disks to hold the 2 gigabyte file.) We have also plotted the curve for a 200 megabyte file with a 32K buffer using our "present day" parameters, for comparison. In the new scenario, striping speedups are 3 to 1 or more, and clearly striping pays off.

Our results show that striping has limitations and must be used with care. In all cases, the speedup obtainable by striping tends towards a constant value as the number of disks increases. Striping is very sensitive to parameters such as available memory buffer space and the amount of processing that must be performed on the data. This means that selecting the optimum number of disks, n, to stripe for an application is difficult, especially if the same data file is going to be used in different

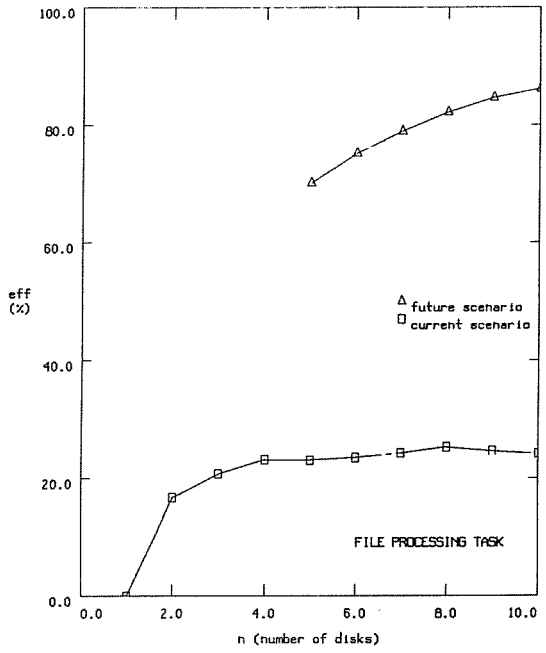

Figure 6

ways.

The striping enhancements can improve performance and at least some of them should be included in a general purpose operating system facility. In addition to increasing effectiveness and reducing service time, matching disks could actually simplify system implementation (each subblock would have the same address on the disk). Ordering the blocks can improve performance, but in this case the application must inform the system where blocks should be placed. Disks with immediate reading capability, if used with care, can provide significant reductions in IO service time and still allow for effective striping.

As mentioned in the introduction, a reliable striping system must include error recovery and detection codes in order to avoid total data loss in a group. Probably the best strategy is to have detection codes within each disk unit, so that a failing unit can be pinpointed. Then the correcting codes can be stored in redundant disk drives.

For example, suppose that we want to correct a single failure. Then we only need one extra disk with parity bits. For each stored block, there will be a corresponding subblock in the extra disk. The first bit in the subblock will be the parity bit for the set of bits that constitute the first bit of the other subblocks. The second bit will be the parity for the second set of bits, and so on. When a failure occurs, the codes internal to each disk will be used to identify the subblock that has been lost. Then the parity block can be used to reconstruct the missing subblock. If having delays in the correction phase is tolerable, then the parity block does not have to be read until an error is detected. Otherwise, it has to be read concurrently with the other subblocks. (Note that if the lost subblock cannot be identified, more than a single parity subblock is required.) To correct more failures, this example can be generalized using well known erasure correcting codes [1].

References

[1] E. Berlekamp, "*Algebraic Coding Theory*," McGraw-Hill, 1968, pp. 229-231.

[2] H. Boral, D.J. DeWitt, "Database Machines: An Idea Whose Time Has Passed? A Critique of the Future of Database Machines," *Database Machines*, H.-O Leilich, M. Missikoff, ed., Springer-Verlag, 1983, pp. 166-187.

[3] "*RA80 Disk Drive User Guide*," Digital Equipment Corporation, 1981.

[4] "*Peripherals Handbook*," Digital Equipment Corporation, 1981.

[5] "*RA81 Disk Drive User Guide*," Digital Equipment Corporation, 1982.

[6] D.J. Dewitt, R.H. Katz, F. Olken, L.D. Shapiro, M.R. Stonebraker, D. Wood, "Implementation Techniques for Main Memory Database Systems," *SIGMOD Record*, Vol.14,#2,1984.

[7] H. Garcia-Molina, R.J. Lipton, J. Valdes, "A Massive Memory Machine," *IEEE Transactions on Computers*, Vol. C33, No. 5, May 1984, pp. 391-399.

[8] J. Gray, "What Difficulties are Left in Implementing Database Systems," Invited Talk at *SIGMOD Conference*, San Jose, CA.,May 1983.

[9] P. Honeyman, personal communication.

[10] M.Y. Kim, "Parallel Operation of Magnetic Disk Storage Devices: Synchronized Disk Interleaving," *Proc. of the Fourth Int'l Workshop on Database Machines*, March, 1985, pp. 299-329.

[11] D. Mason, personal communication.

[12] K. Salem, H. Garcia-Molina, "Disk Striping (Full Version)," Technical Report 332, Department of Electrical Engineering and Computer Science, Princeton University, December 1984.

[13] H.J. Siegel, F. Kemmerer, M. Washburn, "Parallel Memory System for a Partitionable SIMD/MIMD Machine," *Proc. 1979 Int'l. Conference on Parallel Processing*, pp. 212-221.

[14] G. Wiederhold, "*Database Design*," McGraw-Hill, 1977, Chaps. 2,3.