Technical Memorandum 45312-891206-01TM, AT&T, Bell Laboratories, Department 45312, Holmdel, NJ, December 1989.

Reliability and Performance of Parallel Disks

Azer Bestavros *
Harvard University
Cambridge, MA

Danny Chen
AT&T Bell Laboratories
Holmdel, NJ

Wing Wong
AT&T Bell Laboratories
Holmdel, NJ

March 1990

Abstract

Recently, parallel disk systems have emerged as a potential solution for achieving ultra-high capacity, performance and reliability at a reasonably low cost. This emerging technology is likely to have a great impact on the market of data-storage devices for mainframe computers and large database systems [Zachary:89]. The basic idea is to disperse (or *stripe* data blocks over a number of small and cheap physical disks. Accessing these disks in parallel guarantees a linear speedup in the transfer rates. Moreover, introducing storage redundancy promises an ultra-high level of availability that is impossible to achieve using usual approaches. This added redundancy can be exploited to further improve the system performance by reducing the seek and latency delays of the overall system.

In this paper, we investigate a number of issues related to the design of systems of parallel disks. We start with a detailed performance analysis of these systems. In this respect, we obtain expressions for the different delays incurred; namely the queuing, seek, latency, transfer and overhead delays. Next, we consider the reliability requirements of these systems. We show that by adding a small amount of redundancy, the reliability of parallel disks can be made very high. Next, we present a number of potential hardware architectures; namely, bus oriented, sliced memory, and switching network organizations. Finally, we discuss a number of design decisions concerning redundancy, synchronization and grain size.

*This work was done while the author was with AT&T Bell Laboratories, August 1989.

1 Introduction

Recently, parallel disk systems have emerged as a potential solution for achieving ultra-high capacity, performance and reliability at a reasonably low cost. This emerging technology is likely to have a great impact on the market of data-storage devices for mainframe computers and large database systems [Zachary:89]. Interest in parallel disk systems has been spurred by the sharp decline in the price of small hard disk drives. The basic idea is to use a large number of *inexpensive* small disks to attain very large capacities. The relatively low cost per megabyte of storage on these disks,[1] combined with their better volumetric and power efficiencies [Patterson:89], make them very attractive when compared to larger disks. By responding to a larger number of requests concurrently, the overall throughput of the system is increased. Moreover, by accessing a number of disks in parallel in response to a single request, the response time per request may be decreased. Finally, by introducing a relatively low level of redundancy, the availability/reliability of the system can be made astronomically high. This added redundancy can be exploited to further improve the system performance by reducing the seek and latency delays and enhancing concurrency. The enthusiasm for parallel disk systems reflects a growing concern that perhaps too much effort has been spent on building ever-faster processors at the expense of building better storage devices and other types of peripheral equipment.

In this paper, we investigate a number of issues related to the design of systems of parallel disks. We start with a detailed performance analysis of these systems. In this respect, we obtain expressions for the different delays incurred; namely the queuing, seek, latency, transfer and overhead delays. Next, we consider the reliability requirements of these systems. We show that by adding a small amount of redundancy, the reliability of parallel disks can be made very high. Next, we present a number of potential hardware architectures; namely, bus oriented, sliced memory, and switching network organizations. Finally, we discuss a number of design decisions concerning redundancy, synchronization and grain size.

2 Performance of Parallel Disks

The response time of a disk access consists of two components: the *queuing time* and the *service time.* The queuing time, t_Q, is the time it takes from when an access is presented to the disk server to when service begins. This is basically the time spent waiting for the controller and/or the disk to become available. The service time, t_S, is the time it takes the server to complete the access.

There are two possible ways parallelism can be exploited to reduce the response time of a disk access. On the one hand, by having more servers[2] in the system, we can expect the queuing time to be reduced. We call this kind of parallelism *external.* On the other hand, by having more *physical disks* per server, we can expect the service time to be reduced. We call this kind of parallelism *internal.* Any parallel disk system will, most probably, incorporate both internal and external parallelism. Figure 1 illustrates a system where both external and internal parallelism are used.

In this section we analyze the potential performance gain in both the service time and queuing time when parallelism is used.

[1] Roughly 5-10 Dollars per Megabyte for 3.5-inch disks (*e.g.* Conners CP3100) compared to 20-45 Dollars per Megabyte for the 10.5-inch and the 14-inch disks (*e.g.* DEC RA-81, IBM 3380).

[2] groups of *logical disks*

Figure 1: Internal Parallelism (Striping) versus External Parallelism

2.1 Disk Service Time

The service time, t_S, consists of four components. The seek time, t_s, is the time it takes for the head to move to the appropriate track. The latency time, t_l, is the time it takes for the requested sector to come underneath the disk head. The transfer time, t_t, is the time needed to transfer the requested block of data to the memory. Finally, the overhead time, t_o, represents any delays imposed by the controller to process the access. The service time, t_S, is given by:

$$t_s = t_s + t_l + t_t + t_o$$

The relative contribution of these components might differ by orders of magnitude. For instance, the overhead time might be as small as few microseconds whereas the seek and latencies might be as large as few milliseconds. This, however, might not always be the case. Thus, we will consider every one of these components separately. For a given system, some of these components might be negligible.

The seek, latency, transfer and overhead delays are well defined for single disk systems. The service time is simply their sum. For parallel disk systems, this is not the case. In a way, these delays can be interleaved and/or correlated. In the analysis that follows, we consider each one of these delays separately. This is a simplification that would allow us to obtain an *upper* bound on the expected service time by adding up the expected seek, latency, transfer and overhead delays. This is an acceptable simplification since, in most of the cases, one of these (usually the seek time) will dominate.

2.1.1 Seek Time:

The seek time of an access is directly proportional to the distance the head will have to travel to service that access. For simplicity we will assume that the constant of proportionality is 1. In a typical system, the relation between the seek time and the traveled distance is non-linear. For instance, moving across 10 tracks does not take 10 times the time it takes to move 1 track. For example, Figure 2 shows the seek characteristics of a DEC RP06 system. The time to move across 1, 10, 100, and 200 cylinders is measured to be about 5, 8, 20, and 25 milliseconds respectively. For simplicity, however, we will assume a linear direct relationship. Furthermore, we will assume that the head position is a continuous variable in the range $[0, D]$ (another simplifying assumption) and that the disk accesses are independent and uniformly distributed over that interval. This latter assumption is not realistic. Data accesses tend to be correlated and non-uniformly distributed over the disk cylinders (due to locality of references, hot spots, arm scheduling, and/or file system optimizations [McKosick:83]). Nevertheless, these assumptions are justified since the effect of non-uniformity and dependence of accesses is to reduce the average seek time. In a sense, these assumptions make our analysis a worst case analysis.

Figure 2: Measured seek time of a DEC RP06 system

Single Disk[3]

Consider a system consisting of a single disk. Let s be the random variable denoting the position of the last access and r be the random variable denoting the position of the incoming access. Both s and r are uniformly distributed in the range $[0, D]$. Let x be the random variable denoting the seek distance for the incoming access.

$$x = |r - s|$$

$$\text{Prob}(x \geq z) = \text{Prob}(r - s \geq z) + \text{Prob}(s - r \geq z) = (1 - \frac{z}{D})^2$$

$$E(x) = \int_0^D (1 - \frac{z}{D})^2 \cdot dz = \frac{D}{3}$$

Thus, for a single disk system, we would expect an average of one third of the maximum seek for every disk access in a single disk system.

When deriving the above estimate, we have assumed that the head remains at the cylinder it has last accessed. A potential gain can be obtained if the head is automatically returned to the central cylinder as soon as an access terminates. This would result in reducing the expected seek time from $D/3$ to $D/4$ (8.2 percent).

Multiple Disks

Consider a system consisting of n disks where an access can be serviced if at least m (where $m \leq n$) of these disks respond. In this section, we will derive an expression for the expected seek time of such a system. Notice that the single disk case is just a special case, where $n = m = 1$. The result we will get can be used to obtain the expected seek time for the different layout approaches.

Let t be the random variable denoting the seek time for the aforementioned system. Since only m out of the n disks are needed to complete the access, it is obvious that t is the maximum seek time of the set consisting of the minimum m seeks out of the n disks.

$$\begin{aligned} \text{Prob}(t \geq z) &= \text{Prob}(\text{At least } n - m \text{ of the seeksare } \geq z) \\ &= \sum_{r=n-m+1}^{n} \left(\begin{array}{c} n \\ r \end{array} \right) P^r (1 - P)^{n-r} \end{aligned}$$

where P, the probability of a single disk having a seek of z or more is given by:

$$P = (1 - \frac{z}{D})^2$$

To compute the expected seek time, we proceed as follows:

$$\begin{aligned} t_s &= E(t) \\ &= \int_0^D \left[\sum_{r=n-m+1}^{n} \left(\begin{array}{c} n \\ r \end{array} \right) P^r (1 - P)^{n-r} \right] \cdot dz \\ &= \sum_{r=n-m+1}^{n} \left(\begin{array}{c} n \\ r \end{array} \right) \int_0^D P^r (1 - P)^{n-r} \cdot dz \end{aligned}$$

[3] The seek time for a single disk is a well known result. We derive it here for completeness.

$$
\begin{aligned}
dz &= -\frac{D}{2\sqrt{P}} \cdot dP \\
t_s &= \sum_{r=n-m+1}^{n} \left(\begin{array}{c} n \\ r \end{array} \right) \int_1^0 -\frac{D}{2\sqrt{P}} P^r (1-P)^{n-r} \cdot dP \\
&= \sum_{r=n-m+1}^{n} \left(\begin{array}{c} n \\ r \end{array} \right) \frac{D}{2} \int_0^1 P^{r-\frac{1}{2}} (1-P)^{n-r} \cdot dP \\
&= \sum_{r=n-m+1}^{n} \frac{D}{2} \frac{\Gamma(n+1)}{\Gamma(n-r+1)\Gamma(r+1)} \frac{\Gamma(r+0.5)\Gamma(n-r+1)}{\Gamma(n+1.5)} \\
&= \frac{D}{2} \frac{\Gamma(n+1)}{\Gamma(n+1.5)} \sum_{r=n-m+1}^{n} \frac{\Gamma(r+0.5)}{\Gamma(r+1)}
\end{aligned}
$$

Notice that the above expression reduces to $D/(2n+1)$ for $n \geq m = 1$. This corresponds to the case where any *one* of the n disk accesses is enough to complete the request. This is the case for read requests in systems where n replicas of each block of data are maintained on different disks.[4] The case where $n = m$ corresponds to the case where *all* n disk accesses are necessary to complete the request. This is the case for write accesses in systems where a data block is spread or replicated over n disks. For a single disk system, where $n = m = 1$, the above expression reduces to $D/3$.

2.1.2 Latency Time:

Let R be the time of one complete disk revolution. Thus, R, is the maximum latency time for any disk access. We will assume that disk accesses are independent and uniformly distributed over the sectors.

Single Disk[5]
Consider a system consisting of a single disk. Let y_i be the continuous uniformly distributed random variable, $0 \leq y_i \leq R$, denoting the latency time for an access.

$$\text{Prob}(y_i \geq z) = 1 - \frac{z}{R}$$

$$t_l = E(y_i) = \int_0^R (1 - \frac{z}{R}) \cdot dz = \frac{R}{2}$$

Thus for a single disk system, we would expect a average latency delay of one half of the rotation time.

Multiple Disks
Consider a system consisting of n disks where an access can be serviced if at least m (where $m \leq n$) of these disks respond. In this section, we will derive an expression for the expected latency time of such a system. Notice that the single disk case is just a special case, where $n = m = 1$. The result we will get will be used to obtain the expected latency time for the different striping approaches.

[4] These systems are called *mirrored* or *shadowed* systems.
[5] The latency time for a single disk is a well known result. We derive it here for completeness.

Let t be the random variable denoting the latency time for the aforementioned system. Since only m out of the n disks are needed to complete the access, it is obvious that t is the maximum latency time of the set consisting of the minimum m latencies out of the n disks.

$$\begin{aligned}\text{Prob}(t \geq z) &= \text{Prob}(\text{At least } n-m \text{ of the latencies are } \geq z) \\ &= \sum_{r=n-m+1}^{n} \left(\begin{array}{c} n \\ r \end{array} \right) P^r (1-P)^{n-r}\end{aligned}$$

where P, the probability of a single disk having a latency of z or more, is given by:

$$P = (1 - \frac{z}{R})$$

To compute the expected latency time, we proceed as follows:

$$\begin{aligned} t_l &= E(t) \\ &= \int_0^R \left[\sum_{r=n-m+1}^{n} \left(\begin{array}{c} n \\ r \end{array} \right) P^r (1-P)^{n-r} \right] \cdot dz \\ &= \sum_{r=n-m+1}^{n} \left(\begin{array}{c} n \\ r \end{array} \right) \int_0^R P^r (1-P)^{n-r} \cdot dz \\ dz &= -R \cdot dP \\ t_l &= \sum_{r=n-m+1}^{n} \left(\begin{array}{c} n \\ r \end{array} \right) \int_1^0 -R P^r (1-P)^{n-r} \cdot dP \\ &= \sum_{r=n-m+1}^{n} \left(\begin{array}{c} n \\ r \end{array} \right) D \int_0^1 P^r (1-P)^{n-r} \cdot dP \\ &= \sum_{r=n-m+1}^{n} R \frac{\Gamma(n+1)}{\Gamma(n-r+1)\Gamma(r+1)} \frac{\Gamma(r+1)\Gamma(n-r+1)}{\Gamma(n+2)} \\ &= R \frac{m}{(n+1)} \end{aligned}$$

2.1.3 Transfer Time:

Assume that the block size in bits is B and the baud rate in bps is b. Furthermore, assume that the transfer is done in parallel using m disks. It is easy to show that the transfer time t_t is given by:

$$t_t = \frac{1}{m} \frac{B}{b}$$

The above expression is meant to emphasize the fact that the transfer time is directly proportional to the block size and inversely proportional to the baud rate. It does not take into consideration other important factors like possible bottlenecks in the system that might result in a larger transfer time. In particular, we have assumed that there is no contention for resources (*e.g.* controller and/or bus). For instance, if the transfer is done through a bus, then the transfer time is non-deterministic and highly dependent on the traffic on that bus.

2.1.4 Overhead Time:

In many systems, the disk controller performs some computations before, while or after accessing the data. These computations are usually performed on chunks of data. These chunks are usually a small fraction of the total block size. Let c denote the size of these chunks. Assuming the transfer of these chunks and the processing by the controller to be pipelined activities, and assuming that the processing time is smaller than the transfer time, it is obvious that an overhead of at least the transfer time of one chunk is experienced per block transfer. Thus the overhead time is proportional to c. In all the analysis that follows, we will assume that the overhead time is negligible.

2.2 Queuing Time

Let the total number of servers (logical disks) in the system be s. Assume that disk access requests follow a Poisson process with rate λ and that the service time for each request is exponentially distributed with a mean $t_s = 1/\mu$.

If any server can be used to service any pending request, then the system can be viewed as an M/M/s process. In this process, the queuing time, t_Q, can be derived. Using the analysis in [Bertsekas:87], we get:

$$
\begin{aligned}
t_Q &= p_0 (\frac{\lambda}{\mu})^s \frac{1}{s!(1-\frac{\lambda}{s\mu})} \frac{1}{(s\mu - \lambda)} \\
p_0 &= \left[\sum_{i=0}^{s-1} \frac{(\frac{\lambda}{mu})^i}{i!} + \frac{(\frac{\lambda}{mu})^s}{s!(1-\frac{\lambda}{s\mu})} \right]^{-1}
\end{aligned}
$$

Obviously, the larger the value of s the smaller the queuing time will be. This performance gain, however, depends on the value of λ/μ.

2.3 Response Time

The response time is simply the total delay experienced from when a request is submitted to when the service is completed. The expected total delay, t_D, is given by:

$$t_D = t_S + t_Q$$

where, t_S and t_Q are the service time and queuing time derived before.

If the total number of disks in the system is limited to N (due to, say, cost constraints) and the number of disks per server is n, it follows that $s = N/n$. A larger n would mean a higher level of internal parallelism, but, a lower level of external parallelism. Obviously, for a given set of parameters (that is $b, B, \lambda, D, ...$ *etc.*) and a given type of workload (read/write mix), an optimal balance between internal and external parallelism can be determined. For instance, Figure 3 illustrates that point. As the number of servers increases, the queuing time decreases whereas the service time increases. The decrease in the queuing time is rather sharp at the beginning but tends to slow down as the number of servers increases (they will be idle anyway!) On the contrary, the service time deteriorates (increases) as the number of servers increases (due to the decrease in internal parallelism). This deterioration depends on the technique used to achieve internal parallelism and on the relative contribution of the seek, latency, transfer, and overhead times to the service time.

Figure 3: Response time versus the number of servers

3 Reliability of Parallel Disks

The reliability of computing systems is of obvious importance to all who expect to benefit from, or who are in any way dependent on, the correct and continuous service that these system are supposed to provide. The reliability requirements [Rennels:84] of different environments may differ enormously. One extreme is the case of critical applications where no maintenance or manual repair activities are feasible, and incorrect results or large delays are completely unacceptable. In the other extreme are environments where data loss penalty is comparatively low and delays are tolerable. Another type of reliability requirement is that of on-line databases where safeguarding the data held is usually much more important than providing continuity of access [Avizienis:76]. In brief, different systems might have different reliability and availability requirements.

Due to their mechanical nature, disk systems have been always the most unreliable and fragile components in computing systems. This is why elaborate fault-avoidance and fault-tolerance [Randell:79] techniques are usually used to increase their reliability and availability. Fault-avoidance enhances the overall system's reliability by using higher quality components, better assembly, extensive testing, regular maintenance, *etc.* to minimize the probability of a malfunction. Fault-tolerance, on the other hand, seeks the same goal by using protective redundancy to recover from the effect of malfunctions. In a sense, malfunctions are expected to happen, but, if they do, their effects are automatically counteracted by the redundancy.

Fault-avoidance cost is usually attributed to the quality of the system components. It grows superlinearly (usually exponentially) with the required level of reliability. Fault-tolerance cost, on the other hand, is usually attributed to the level of redundancy and overhead in the system. It usually grows linearly with the required level of reliability. Fault-tolerance does not entirely eliminate the need for reliable components; instead it offers the option to allocate part of the resources to include redundancy. As a matter of fact, balancing the allocation of resources between fault-avoidance and fault-tolerance is required to achieve the best reliability/cost ratio. For lower levels of fault coverage, the fault-avoidance approach is usually more cost effective. For higher levels of fault coverage, fault-tolerance becomes more effective. Figure 4 illustrates (qualitatively) the balance between quality and redundancy.

3.1 Terminology

A disk failure results from an abnormal physical change of the hardware (*e.g.* media failure). A fault is any abnormal behavior of the system (*i.e.* not in accordance with the system's specifications). A fault might be *hard* (permanent) or *soft* (transient). Hard faults are usually caused by failures, whereas soft faults are caused by environmental factors (*e.g.* noise). A fault is manifested through the occurrence of errors. Disk manufacturers classify errors as either recoverable, unrecoverable or undetectable. By adding redundancy to the system, a larger class of errors can be made recoverable. For example, by using a single Error Correcting Code (ECC), single bit errors can be corrected using the added redundancy, making these errors recoverable. When the added redundancy is not enough to recover an error, it might be enough to detect it. In this case the error is said to be unrecoverable. Unrecoverable errors are catastrophic since they mean loosing data. It might be also possible for the added redundancy not to be able to detect errors or to miscorrect them. In this case the error is said to be undetectable. Undetectable errors are, by far, the most difficult to deal with.

Figure 4: Quality and Redundancy Trade-off

3.2 Reliability Computation

When internal parallelism is used, a data block is scattered over a number of disks. Assume that the Mean Time To Fail (MTTF) for a single disk is T_f, and the Mean Time To Repair (MTTR) for a single disk is T_r. Furthermore, assume that the data blocks are scattered over n disks and that only m out of these are necessary to reconstruct the data back. Let p_k denote the probability of the system having k failures. Assuming the system can be approximated by an M/M/1 process with a finite population of n, we have:

$$\begin{aligned}
p_0 &= \frac{1}{T_r} p_1 \\
p_1 &= \frac{1}{T_r} p_2 + \frac{n}{T_f} p_0 \\
p_2 &= \frac{1}{T_r} p_3 + \frac{n-1}{T_f} p_1 \\
\ldots &= \ldots \\
p_k &= \frac{1}{T_r} p_{k+1} + \frac{n-k+1}{T_f} p_{k-1} \\
\ldots &= \ldots \\
p_n &= \frac{1}{T_f} p_{n-1}
\end{aligned}$$

The solution to the above system[6] is given by:

$$\begin{aligned}
p_k &= p_0 \rho^k \frac{n!}{(n-k)!} \\
p_0 &= \left[\sum_{k=0}^{n} \rho^k \frac{n!}{(n-k)!} \right]^{-1} \\
\rho &= \frac{T_r}{T_f}
\end{aligned}$$

Obviously, for $k \leq (n-m)$ any data blocks are retrievable. However, for $k > (n-m)$ the system fails. Thus:

$$\text{Prob(System Failure)} = \sum_{k=n-m+1}^{n} p_k$$

3.2.1 Single Disk System:

For a system consisting of a single disk, we have $m = n = 1$. Substituting in the above expression we get:

$$\text{Prob(Single Disk System Failure)} = \frac{\rho}{1+\rho}$$

[6] Refer to [Bertsekas:87] for a full derivation of the solution.

3.2.2 Non-redundant Multiple Disk System:

For a system consisting of a multiple disks, with no added redundancy, we have $m = n$. Substituting in the above expression we get:

$$\text{Prob}(\text{Non} - \text{redundant System Failure}) = \sum_{k=1}^{n} p_k = 1 - p_0$$

To compare this with the single disk system, we compute the ratio between the probability of a failure in both cases as follows:

$$\begin{aligned} Q &= \frac{\text{Prob}(\text{Non} - \text{redundant System Failure})}{\text{Prob}(\text{Single Disk System Failure})} \\ Q &= \frac{1 - \left[\sum_{k=0}^{n} \rho^k \frac{n!}{(n-k)!}\right]^{-1}}{\frac{\rho}{(1+\rho)}} \end{aligned}$$

The value of ρ is typically very small. For instance, for a MTTF of one year (this is a typical conservative estimate) and a MTTR of few hours, the value of ρ would be approximately 0.0001. Assuming that $n\rho$ is much smaller than 1 and neglecting higher order terms, we get:

$$\begin{aligned} Q &= \left\{1 - \left[1 + n\rho + n(n-1)\rho^2 + \ldots\right]^{-1}\right\} \frac{(1+\rho)}{\rho} \\ &\approx \left\{1 - \left[1 + n\rho\right]^{-1}\right\} \frac{(1+\rho)}{\rho} \\ &\approx \frac{n\rho}{(1+n\rho)} \frac{(1+\rho)}{\rho} \\ &\approx n \end{aligned}$$

Thus, for a system of multiple disks with no added redundancy, the probability of losing data increases *linearly* with the number of disks in the system. For large values of n this level of reliability becomes unacceptable. For instance, with $n \approx 32$, the MTTF drops from one year (for a single disk system) to few days !

3.2.3 Redundant Multiple Disk System:

Assume that only m of the n disks are needed to reconstruct the data. Thus:

$$\text{Prob}(\text{SystemFailure}) = \sum_{k=n-m+1}^{n} p_k$$

Now, computing the ratio Q, we get:

$$\begin{aligned} Q &= \frac{\text{Prob}(\text{Redundant System Failure})}{\text{Prob}(\text{Single Disk System Failure})} \\ &= \frac{\left[\sum_{k=0}^{n} \rho^k \frac{n!}{(n-k)!}\right]^{-1} \sum_{k=n-m+1}^{n} \rho^k \frac{n!}{(n-k)!}}{\frac{\rho}{(1+\rho)}} \end{aligned}$$

Assuming ρ to be small enough, we get the following approximation:

$$\begin{aligned} Q &\approx \frac{(1+\rho)}{\rho} \frac{\rho^{n-m+1} \frac{n!}{(n-m+1)!}}{(1+n\rho)} \\ &\approx n(\rho n)^{n-m} \end{aligned}$$

Notice that setting $n = m$, we get $Q = n$ as before. For $n = 9$, $m = 8$, and $\rho = 0.001$ (very conservative) the ratio Q evaluates to 0.081, making the redundant system much more reliable than the single disk system. By adjusting the amount of redundancy (that is the value of m with respect to n), any level of reliability can be guaranteed.

4 Hardware Organization

An issue of major importance in designing a system of parallel disks is how it would fit in the overall architecture of the computing system. In this section, we consider a number of alternatives. For any proposed organization, two issues are usually of utmost importance: scalability and transparency. In addition, the cost, the feasibility, and the effect on performance of any proposed architecture are also major concerns.

4.1 Bus Oriented Organization

This is the simplest and most straightforward organization. The server as well as the disks and memory are hooked to a bus. Direct Memory Access (DMA) is used to transfer the data to/from every one of the disks from/to the memory unit. The bus is time multiplexed amongst the different disks. Figure 5 illustrates this simple organization.

Figure 5: Bus Oriented Organization

The main problem with the above organization is that it does not scale up. With a larger number of disks, the bus will eventually pose a serious bottleneck due to its limited bandwidth.

4.2 Sliced Memory Organizations

In this organization (shown in Figure 6), requests are issued to the disk system using the *disk request bus*. Instead of transferring the data directly to/from the main memory and thus creating an unnecessary bottleneck, the data is stored in a distributed memory (one memory bank per disk), that is directly available to the processor(s) through the *memory bus.* Each one of the memory banks is basically a dual-ported RAM that can be accessed from the disks as well as from the memory bus. This is very similar to interleaved memory designs. As a matter of fact, requests to the memory banks can be interleaved to achieve higher throughput. It is also possible to augment this *sliced* memory with a *contiguous* single-ported memory (as shown in Figure-6) to act as a shared main memory for the processors. This might be desirable, since dual-ported memories are usually more expensive and their access by the processors (as a normal main memory) might interfere with ongoing disk I/O operations.

One way of looking at the distributed memory is to envision it as being *sliced* amongst the different disks. This is shown in figure 7. A request to read (write) a data block is basically a request for each disk to read (write) one of the slices. The different memory banks serve as the main memory of the system. Any memory request should hit no more than one of these memory banks, or else, it should be available in the contiguous memory unit (if any).

A major advantage of the sliced memory organization is its scalability. Adding more disks/banks does not introduce any bottlenecks. This is due to the fact that the memory is sliced (partitioned) and not shared. This organization, however, is not scalable in the number of processing units in the system. In the uniprocessor case, neither the disk request bus, nor the memory bus pose any bottlenecks. In a multiprocessor configuration, however, both the disk request bus and the memory bus are potential bottlenecks since they become shared resources. Cashing and bus arbitration become necessary. A nice aspect of a sliced memory organization is its transparency. Ideally, an operating system need not be aware of the fact that the memory is sliced or that a number of disks are involved in every disk access. Due to this transparency, however, flexible reconfiguration of the system becomes extremely difficult.

4.3 Switching Network Organization

In this organization, the different servers (or processors) are allowed to access any subset of the available disks through the use of a switching network (circuit or packet switching). Basically, the switching network allows a number of designated disks to be connected to the bus of a processor upon its request. This organization is illustrated in Figure 8. Notice that, the data transfer can be done through Direct Memory Access (DMA).

The main advantage of this organization is its flexibility in terms of the availability of the different groups of disks to the different servers. In a sense, the operating system can adjust the level of reliability / performance on demand. The main problem with such an organization is the design of the switching network. It does not scale nicely with the number of processors and/or disks. The complexity of this switch as well as its cost and delay grow rapidly with the number of disks and servers. The switching network organization solves the problem of potential contention over the disk request bus in a multi-processor environment, but, does not solve the potential contention over the memory bus due to competing DMA accesses between the disks and memory.

Figure 6: Sliced Memory Organization

Figure 7: Memory map amongst the different slices

Figure 8: Switching network organization

4.4 Switching Network / Sliced Memory Organization

In this organization (see Figure 9), a cross-bar switch is used to connect a server's bus to any subset of disks. Every subset of disks is connected to a memory bank (dual-port RAM). Transfer to/from the data banks is done using Direct Memory Access (DMA). The number of disks per group is limited by the capacity of the group bus, whereas the number of groups is limited by the complexity and cost of the communication switch. This organization is suitable for systems consisting of a large number of disks. For instance a system of 128 disks can be organized into 16 groups each consisting of 8 disks.

5 Software Organization

As we mentioned before, one key reason for building a system of parallel disks is to create a server (a logical disk) that has a very high data transfer rate. This can be achieved by ganging together a set of physical disks; *striping* data blocks across them; and transferring data to/from them in parallel. The performance (namely the service time) of such an organization is highly dependent on a number of design decisions. In this section, we discuss these in details. We make the assumption that data blocks are striped across n disks and that reading m out of these ($m \leq n$), is enough to reconstruct the original data.

5.1 Level of Synchronization

There are three possible assumptions concerning the level of synchronization between the physical disks of a given server. In the first, both the head movements and the cylinder rotations are synchronized. We call such a system *fully synchronous.* In the second, the head movements are synchronized while the cylinder rotations are not. We call such a system *head synchronous.* In the third, neither the head movements nor the cylinder rotations are synchronized. We call such a system *asynchronous.*

In the analysis that follows, we compute the expected seek and latency delays for different synchronization assumptions. The sum of these expected values imposes an *upper* bound on the expected delay of the system. It is an upper bound due to the existence of nonlinear operators (like *max* and *min*). For instance, if m pieces out of n are sufficient to reconstruct the original data, then the expected delay due to seek and latency should be the maximum of the smallest m out of n sums of the seek and latency random variables. Computing the expected seek and latency delays separately and then adding them is an acceptable simplification since, in most of the cases, one of these (usually the seek time) will dominate.

5.1.1 Fully Synchronous Systems:

If the physical disks of a server are fully synchronized and if the different stripes of a data block are located on the same track and sector on every disk, then the average seek and latency times for that server are basically the same as for a single disk system. From our previous analysis, we get:

$$t_s = \frac{D}{3}$$
$$t_l = \frac{R}{2}$$

The above estimates do not take into consideration any optimizations that might be possible due to the determinism associated with the synchronous seek and rotation of the physical disks. For instance,

Figure 9: Switching network / Sliced Memory Organization

if $n = 2$ and $m = 1$, that is any one stripe out of two is sufficient, then by laying out the two stripes 180 degrees apart, the expected latency time of a read access can be reduced from $0.5R$ to $0.25R$. Notice, however, that for a write access, the latency actually increases from $0.5R$ to $0.75R$. The same technique can be used to reduce the read seek time by placing the stripes on different cylinders of the physical disks. For general values of n and m, and for a given read / write mix, the optimization of the combined seek and latency delay might not be straightforward.

5.1.2 Seek Synchronous Systems:

If the physical disks of a server are seek synchronized and if the different stripes of a data block are located on the same track on every disk, then the seek time of the server is the same as that of a single disk, whereas its latency is the maximum of the smallest m latencies out of the n physical disks. From our previous analysis, we get:

$$\begin{aligned} t_s &= \frac{D}{3} \\ t_l &= \frac{m}{(n+1)}R \end{aligned}$$

The same notice we made about possible optimizations in fully synchronous systems can be applied here. In particular, due to the determinism associated with the synchronous head movement, the read seek time can be further reduced.

Seek synchronization can be assumed in two different cases. First, if the hardware is built such that the mechanical movement of the heads of the different disks are synchronized then the seek synchronization assumption is a valid one. Another situation, however, arises when the synchronization is imposed by the software. For instance, if the operating system always stores the different pieces of a file on the same cylinder (or on a small cluster of nearby cylinders) [McKusick:83] then the seek synchronization assumption can be justified. In a sense, we would expect all the drives to be at almost the same corresponding cylinder following any file access. While hardware imposed seek synchronization seems unrealistic,[7] software induced seek synchronization is likely to be common.

5.1.3 Asynchronous Systems:

In this case, the seek time of the server is the maximum of the smallest m seeks out of the n physical disks, whereas the latency is the maximum latency exercised by any one of these m disks. From our previous analysis, we get:

$$\begin{aligned} t_s &= \frac{D}{2}\frac{\Gamma(n+1)}{\Gamma(n+1.5)}\sum_{r=n-m+1}^{n}\frac{\Gamma(r+0.5)}{\Gamma(r+1)} \\ t_l &= \frac{m}{(m+1)}R \end{aligned}$$

[7] For example, a bad track or sector in one of the disks will make the corresponding tracks or sectors on the other disks inaccessible.

5.2 Granularity

When data striping is implemented, an important design decision is the granularity of the stripes. In *fine grain* striping, the size of a stripe is fairly small (one or few bits or words). In *coarse grain* striping, the size of a stripe is fairly large (one or few sectors or tracks).

An advantage of fine grain striping over coarse grain striping is that the overhead time, t_o, for assembling / disassembling the data blocks is small (often negligible). A disadvantage, however, is that no matter how small the required data to be accessed is, *all* of the physical disks have to be accessed. This will affect the achievable level of concurrency.

5.3 Redundancy

Different approaches have been suggested in the literature to incorporate the necessary redundancy in a system of parallel disks.

5.3.1 Shadowing:

This is the simplest and most straightforward approach for increasing the reliability of a system [Bates:85]. The idea is to keep $(r+1)$ replicas of every block of data. Each one of these replicas is kept on a different disk. If one of these disks fails, the system uses one of the redundant copies until the failed disk is repaired. This scheme tolerates up to r failures out of the n disks by an r-fold increase in the required storage.

5.3.2 Parity and Coding:

A major drawback of the shadowing approach is the excessive amount of redundancy needed to protect against failures. Instead of replicating data, redundancy can be added to the system so as to *correct* the erroneous information. In [Patterson:89], [Patterson:88] such an approach for tolerating a single failure in a group of N disks was suggested and termed *N+1 RAID.* The idea is to calculate and store parity information of a group of disks on a bit-per-disk basis. One parity block would be needed for every N blocks across the disks. Any single disk failure can be corrected simply by reading the rest of the disks in the group to determine what bit values of the failed disk would result in proper parity. This approach can be generalized to be able to tolerate more than one failure. In [Gibson:88], a number of coding techniques were devised for use in parallel disk systems.

5.3.3 Information Dispersal Approach:

Recently, Michael O. Rabin [Rabin:87] devised a new algorithm for the secure and fault-tolerant communication and storage of information. The basic idea of this algorithm is to disperse the contents of a data block into n different pieces so that recombining *any* m of these pieces, $m \leq n$, is sufficient to reconstruct the original data block. In [Bestavros:89a] and [Bestavros:89b], we have proposed using IDA in designing systems of parallel disks. Obviously, such an approach would tolerate up to $n-m$ faults. If the size of the data to be dispersed is S then the size of each of the pieces is $(1/m)S$, making the total required storage $(n/m)S$. The major aspect of IDA is that the added redundant information is not identifiable (as is the case with the parity approach). Rather, it is distributed among the data blocks. This makes the different pieces of information in the system *uniform.* That is, there is no distinction between information and

redundancy. In [Bestavros:89b], we showed that IDA is superior to both shadowing and parity, and that it has great potentials for being used in future RAID (Redundant Array of Inexpensive Disks) systems.

6 Conclusion and Future Work

In this paper we have covered a number of issues related to the design of parallel disk systems. We started with an analysis of the performance and reliability of these systems. We showed that a balanced allocation of the available resources between internal and external parallelism and between fault-tolerance and fault-avoidance is crucial. Next, we considered a number of architectural and organizational design decisions and discussed their effect on the system performance.

More work is needed to simulate, prototype and compare the design alternatives we proposed. Also, the impact of using parallel disk systems on the design of controllers and device drivers is yet to be studied.

References

[Avizienis:76] Avizienis A., *Fault tolerant systems,* IEEE transactions on computers December 1976.

[Bates:85] Bates K.H. and TeGrotenhuis M., *Shadowing boosts system reliability,* Computer Design, April 1985.

[Bertsekas:87] Dimitri Bertsekas and Robert Gallager, "Data Networks," *Prentice-Hall, Inc., Englewood Cliffs, NJ,* 1987.

[Bestavros:89a] Azer Bestavros, *A VLSI chip for the real-time information dispersal and retrieval for security and fault-tolerance,* Technical Report, TR-06-89, Dept. of Computer Science, DAS, Harvard University, January 1989. Also, *to appear*, in the Proceedings of the 19^{th} International Conference on Parallel Computing, August 1990.

[Bestavros:89b] Azer Bestavros, *IDA-based Disk Arrays* Technical Memorandum 45312-890707-01TM, Bell Laboratories, July 1989.

[Bitton:89] Dina Bitton, *Arm Scheduling in Shadowed Disks,* COMPCON-89, the Thirty-fourth IEEE Computer Society International Conference, March 1989.

[Gibson:88] Garth Gibson, Lisa Hellerstein, Richard Karp, Randy Katz and David Patterson, *Coding Techniques for Handling Failures in Large Disk Arrays,* Technical Report UCB/CSD 88/477, Computer Science Division, University of California, July 1988.

[Kurzweil:88] Fred Kurzweil, Jr., *Small Disk Arrays – The emerging approach to high performance,* COMPCON-88, the Thirty-third IEEE Computer Society International Conference, March 1988.

[McKusick:83] Marshall McKusick, William Joy, Samuel Leffler, and Robert Fabry, *A Fast File System for UNIX,* Technical Report UCB/CSD 83/147, Computer Science Division, University of California at Berkely, July 1983.

[Ng:89] Spencer Ng, *Some Design Issues of Disk Arrays,* COMPCON-88, the Thirty-fourth IEEE Computer Society International Conference, March 1989.

[Patterson:89] David A. Patterson, Peter Chen, Garth Gibson, and Randy H. Katz, *Introduction to Redundant Arrays of Inexpensive Disks (RAID),* COMPCON-89, the Thirty-fourth IEEE Computer Society International Conference, March 1989.

[Rabin:87] Michael O. Rabin, *Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance* Technical Report, TR-02-87, Department of Computer Science, DAS, Harvard University - April 1987. Also appeared in the Journal of the Association for Computing Machinery, Vol. 36, No. 2, April 1989, pp. 335-348. April 1989.

[Randell:79] Randell B., *System reliability and structuring,* Ch.1 Pp. 1-18 of "Computing systems reliability", Cambridge press, 1979.

[Rennels:84] Rennels D. A., *Fault-tolerant computing - Concepts and examples,* IEEE transactions on computers December 1984.

[Schulze:89] Martin Schulze, Garth Gibson, Randy Katz, and David Patterson, *How Reliable is a RAID?,* COMPCON-89, the Thirty-fourth IEEE Computer Society International Conference, March 1989.

[Zachary:89] Pascal Zachary, *Disk Arrays may reshape storage of computer data,* The Wall Street-Journal (Technology), July 19, 1989.