An Evaluation of Redundant Arrays of Disks using an Amdahl 5890

Peter M. Chen, Garth A. Gibson, Randy H. Katz, David A. Patterson
Computer Science Division, University of California, Berkeley

Abstract. *Recently we presented several disk array architectures designed to increase the data rate and I/O rate of supercomputing applications, transaction processing, and file systems [Patterson 88]. In this paper we present a hardware performance measurement of two of these architectures, mirroring and rotated parity. We see how throughput for these two architectures is affected by response time requirements, request sizes, and read to write ratios. We find that for applications with large accesses, such as many supercomputing applications, a rotated parity disk array far outperforms traditional mirroring architecture. For applications dominated by small accesses, such as transaction processing, mirroring architectures have higher performance per disk than rotated parity architectures.*

1. The I/O Crisis

Over the past decade, processing speed, memory speed, memory capacity, and disk capacity have all grown tremendously:

- Single chip processors have increased in speed at the rate of 40%-100% per year [Bell 84, Joy 85].
- Caches have increased in speed 40% to 100% per year.
- Main memory has quadrupled in capacity every two or three years [Moore 75, Myers 86].

In contrast, disk access times have undergone only modest performance improvements. For example, seek time has improved only about 7% per year [Harker 81]. This imbalanced system growth has already led to many I/O bound supercomputer applications [Kim 87]. If the imbalance is not remedied, Amdahl's Law tells us that much of the astounding processor speedup and memory growth will be wasted [Amdahl 67]. Continued improvement in system performance depends in a large part on I/O systems with higher data and I/O rates.

One way to increase I/O performance is by using an array of many disks [Kim 86, Salem 86]. By using many disks, both throughput (MB per second) and I/O rate (I/O's per second) can be increased. Throughput can be increased by having many disks cooperate in transferring one block of information; the I/O rate can be increased by having multiple independent disks service multiple independent requests. With multiple disks, however, comes lower reliability. According to the commonly used exponential model for disk failures [Schulze 88], 100 disks have a combined failure rate of 100 times the failure rate of a single disk. If every disk failure caused data loss, a 100 disk array would lost data every few hundred hours. This is intolerable for a supposedly stable storage system. To protect against data loss in the face of a single disk failure, some sort of data redundancy must exist.

This paper analyzes the performance of two disk array redundancy schemes. The performance analysis is based on a set of experiments carried out an Amdahl mainframe and disk system. In these experiments, we explore several issues:

- What are the basic differences in throughput and response time between the various redundancy schemes?
- How does changing the size of an I/O request affect performance of the redundancy schemes?
- How does changing the read/write ratio affect performance of the redundancy schemes?

We begin by describing two redundant array organizations and a simple performance model comparing them. We then present the hardware environment, the experiment workload design, the metrics for our results, and the performance results.

We start by running the same workload as the RAID paper [Patterson 88]: request sizes are full stripe or individual blocks; requests are all reads or all writes. In Section 7.1, we analyze an idle system to break down the response time for a basic I/O. Next, in Section 7.2, we attempt to duplicate the assumptions made in the RAID paper of unlimited response time by analyzing a saturated system. In 7.3, we make the experiment more variable by controlling and equalizing the response times.

When we have finished analyzing the RAID paper workload, we begin to use more varied workloads. We again make this transition by changing one aspect of the workload at a time. In 8.1, we remove the assumption of constant request size by using a distribution of request sizes. As our last step in making the experiment more realistic, we allow workloads with both reads and writes.

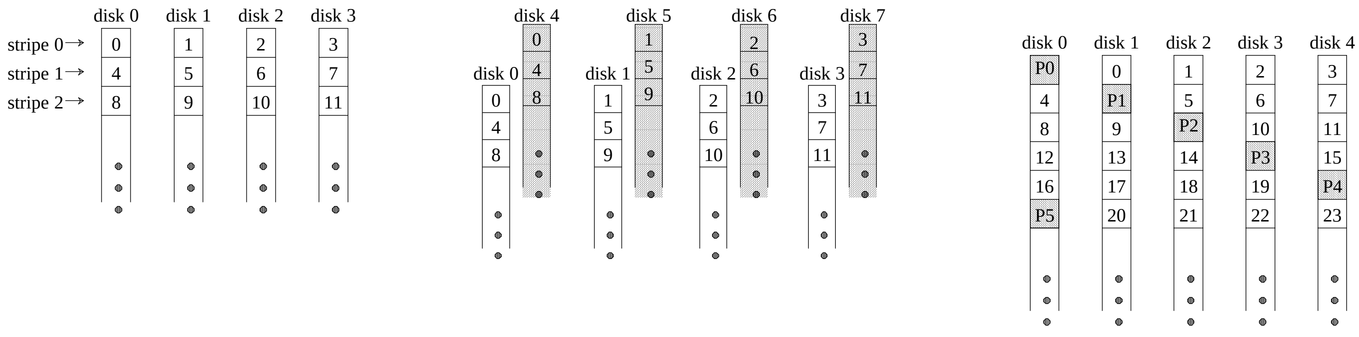
2. Introduction to Redundant Arrays of Disks

In *"A Case for Redundant Arrays of Inexpensive Disks (RAID)"*, henceforth referred to as "The RAID paper" [Patterson 88], Patterson, Gibson, and Katz present five ways to introduce redundancy into an array of disks: RAID Level 1 through RAID Level 5. Using a simple performance model of these five organizations, they conclude that RAID Level 1, *mirrored disks*, and RAID Level 5, *rotated parity*, have the best performance potential. This paper focuses on these two RAID Levels, plus the basic non-redundant RAID Level 0, added to provide a basis of comparison between RAID Levels 1 and 5. Figure 1 shows the data layout in the three redundancy schemes. The rest of this section summarizes the RAID Levels--see [Patterson 88] for more details.

In all organizations, data are interleaved across all disks [Kim 86, Salem 86]. We define a *stripe* of data to be one unit of interleaving from each disk. For example, the first stripe of data in Figure 1 consists of logical blocks 0, 1, 2, and 3. The *storage efficiency*, a measure of the capacity cost of redundancy, is defined to be the *effective (user) data capacity divided by the total disk capacity*. For RAID Level 0, the effective data capacity equals the total disk capacity, so the storage efficiency is 100%.

RAID Level 1, mirrored disks, is a traditional way to incorporate redundancy in an array of disks [Bitton 88]. In RAID Level 1, each datum is kept on two distinct disks: a data disk and a shadow disk. Thus, for RAID Level 1, the effective storage capacity is half the total disk capacity and the storage efficiency is 50%. Reads can be serviced by either the data disk or the shadow disk, but, to maintain consistency, writes must be serviced by both data and shadow disk.

RAID Level 5, rotated parity, incorporates redundancy by maintaining parity across all disks. For example, P0 in Figure 1c is the parity of logical blocks 0, 1, 2, and 3. Parity will have to be updated whenever data is written. If all parity information was kept on one disk, this disk would see many more requests than any data disk. To avoid a bottleneck in accessing the parity information, it is spread over all disks. There are many ways to spread this parity information across disks, but this is not within the scope of this paper. Instead, we have chosen one mapping of



(a) RAID Level 0--non-redundant array

(b) RAID Level 1--mirroring. Shadow disks are shaded.

(c) RAID Level 5--rotated parity. The shaded areas are parity.

Figure 1: Three RAID architectures. Data (logical blocks) are interleaved across multiple disks with various redundancies added. In RAID Level 0 (Figure 1a), no redundancy exists. A stripe of data consists of a logical block from each disk. In RAID Level 1 (Figure 1b), each data disk (disks 0-3) has a shadow disk (disks 4-7). In RAID Level 5 (Figure 1c), parity for each stripe is kept in a parity block. Which physical disk the parity block is kept on is different for different stripes to reduce hot spots for single block writes (see [Patterson 88]).

parity information onto disks. As shown in Figure 1c, parity for stripe 0 is kept on disk 0; parity for stripe 1 is kept on disk 1, and so on. If there are N disks in the array, the storage efficiency is $\frac{N-1}{N}$.

3. A Simple Performance Model

In this section we present the simple performance model used by the RAID paper to compare RAID Levels. First, some terminology is needed. A user request to read or write data is called a *logical* request. A *physical* request refers to a logical request after it has been mapped onto the disk array. Often, due to redundancy information, a physical request will involve more disk blocks than its logical request. A *disk access* refers to one contiguous read or write of one disk. Physical requests result in one or more disk accesses. A logical request that involves all the data in a stripe is called a *full stripe* request. A logical request that involves only part of the data in a stripe is a *partial stripe* request. A special type of partial stripe request is an *individual* request, which is a request to exactly one disk's part of a stripe.

The model in the RAID paper is concerned with the maximum possible throughput of a disk system. The model drives the disk system with four types of logical requests: full stripe reads, full stripe writes, individual reads, and individual writes. To estimate maximum possible throughput, we consider the *efficiency* of a RAID: the number of disk accesses of a logical request divided by the number of disk accesses in its corresponding physical request.

Because RAID Level 0 has no redundant information, the number of disk accesses in a physical request is always the same as in its logical request. Thus RAID Level 0 has an efficiency of 100%, that is, 100% of the disk accesses involve useful data. We normalize throughput of a RAID by defining *relative throughput* of a RAID system running a particular workload as the throughput of that RAID system relative to the throughput of a non-redundant array (RAID Level 0) running the same workload (matching workloads will be described later). The simple model in the RAID paper estimates relative throughput by the fraction of disk accesses involving useful data. For example, if the physical request involves twice as many disks accesses as its corresponding logical request, i.e. the logical to physical mapping doubled the number of disks accesses involved, then 50% of the disk accesses would involve useful data and the simple model would estimate the relative throughput to be 50%.

For all RAID Levels, assuming no failed disks, data can be read without accessing any redundancy information (these experiments do not measure performance when one or more disks are not operational). Because of this, the mapping from logical read requests to physical requests adds no extra disks, and the simple model predicts a relative throughput for RAID Levels 0, 1, and 5 reads of 100%. As mentioned above, RAID Level 0 also has, by definition, a relative throughput of 100% for writes. However, for RAID Levels 1 and 5, physical write requests involve more disk accesses than their logical write requests, and relative throughput becomes less than 100%.

To write data in RAID Level 1, both the data disk and the shadow disk must be written. Thus, a RAID Level 1 physical write request has twice the number of disk accesses as its logical request. The simple model estimates relative throughput at 50% for any size RAID Level 1 write.

For RAID Level 5, both the data disk(s) and the parity information need to be updated. To compute the new parity, some reads may need to be issued. Some of these reads snapshot the image on disk before those blocks are overwritten. We call these *pre-reads*. How much information needs to be read depends on the size of the logical write request. For full stripe writes, no reads are needed, since the new data completely determines the new parity of the stripe. Thus, with an N disk array, a full stripe logical write request involves N-1 disk accesses, and the physical request involves N disk accesses. This leads us to estimate full stripe write relative throughput as $\frac{N-1}{N}$. For partial stripe writes, parity may be computed either by 1) pre-reading the current (before writing) data on the data disk(s) and current parity of the stripe or 2) reading the current data in the rest of the stripe. For example, in Figure 1c, to write logical blocks 0 and 1, we can either 1) pre-read logical blocks 0 and 1 and parity block P0 or 2) read logical blocks 2 and 3. With a partial stripe write of D (less than N-1) data disks, the first method of computing parity involves D+1 disk pre-reads, and the second method involves $N-(D+1)$ disk reads. For an individual block request ($D=1$) the first method is better for $N \geq 4$. With this first method, an individual request, involving one disk access, generates a physical request involving four disk accesses (two to read the current data and current parity and two to write the new data and new parity).

This leads us to estimate relative throughput as 25% for individual block writes in RAID Level 5.

The estimates for full and individual requests for both RAID Levels 1 and 5 are summarized in Figure 2.

4. Introducing Realism

Our overall goal in these experiments is to understand more fully how RAID Levels 1 and 5 perform. This includes exploring aspects of implementation, workload characterization, and performance evaluation. The RAID paper estimated performance based on maximum possible throughput. A specific goal of the experiments described here is to *measure* the performance of RAID Levels 1 and 5 under more variable workloads and to compare this measured performance against the simple performance estimates in the RAID paper. The performance characterization in these experiments differs from the RAID paper in the following areas:

- *Real hardware*: The analysis done in the RAID paper was a purely theoretical analysis. It assumed a constant time for disk accesses and ignored processing overhead. Because these experiments were carried out on an actual machine with disks, they have no need to make any of these simplifying assumptions.
- *Response time*: The only performance metric in the RAID paper was maximum possible throughput. These experiments will measure and control both throughput and response time.
- *Synthetic workload*: Because the analysis in the RAID paper was theoretical and extremely simple, it used unrealistic workloads. These experiments refine the workloads in three ways:
  (1) The RAID paper used an infinite workload, i.e., the disks were fully utilized. These experiments introduce contention, resulting in more realistic (suboptimal) disk utilization.
  (2) The RAID paper workloads had a constant logical request size of either 100% full stripe requests or 100% individual requests. These experiments deal with a distribution of request sizes, including partial stripe accesses and accesses larger than a full stripe.
  (3) The RAID paper workloads were either 100% reads or 100% writes. These experiments explore a range of read/write ratios.

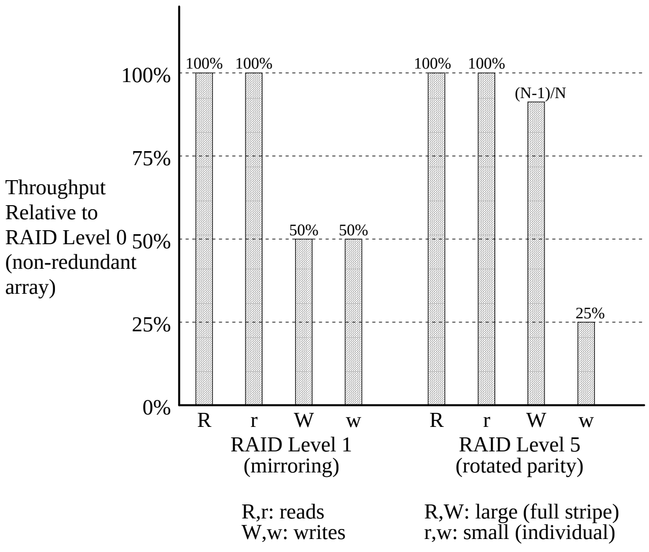


Figure 2: Relative Throughput According to a Simple Model. Shown is the estimated throughput of RAID Level 1 (mirrored RAID) and RAID Level 5 (rotated parity RAID) as a percentage of the estimated throughput of RAID Level 0 (non-redundant RAID) [Patterson 88]. RAID Level 5 large write performance is calculated assuming 11 total disks (*N=11*).

Note that these experiments are not running application programs (benchmarks), but rather an artificially generated distribution of I/O requests (synthetic workload). We choose to use synthetic workloads because they are easier to parameterize than benchmarks, making it possible to explore a range of different user workloads. Synthetic workloads also avoid the access pattern biasing of individual operating systems.

5. Metrics for Comparing RAID Levels

When comparing RAID Levels, we are interested in performance (throughput and response time) and cost. Synthesizing a single metric for comparison is not easy. Because storage efficiency differs between RAID Level 0 (100%), Level 1 (50%), and Level 5 ($\frac{N-1}{N}$), RAID systems with the same user data capacity need different numbers of disks and so have different costs. There are at least two ways to address this issue.

If we hold the total number of disks constant between RAID Levels, then comparing costs is trivial. However, comparing the performance of such RAID systems is tricky because it is unclear how to define an equal workload between systems with different user data capacities. For example, consider two disk systems, A and B, both with the same number of total disks. Suppose that system A has a storage efficiency of 50% and system B has a storage efficiency of 100%. Thus, system B has twice the user data capacity as system A. If A and B receive the same workload, then data in B is accessed half as frequently as data in A. To compensate, we may wish to present B with double the workload that A receives. Unfortunately, it is unclear what constitutes a double workload: double the request rate? Double the request size?

To allow equivalent workloads, these experiments, maintain equal data capacities between RAID Levels. However, with *D* disks of user data, RAID Level 0 needs *D* total disks, RAID Level 1 needs *2D* total disks and RAID Level 5 needs *D+1* total disks. Thus, costs and raw disk bandwidth of the different RAID Levels are no longer equal. We must therefore factor in costs when presenting performance.

One method to factor in costs is to simply present the raw performance and cost separately. For example, in one of the experiments, a RAID Level 1 used 20 disks and yielded a throughput of 20 MB/s. The corresponding RAID Level 5 system used 11 disks and yielded a throughput of 10 MB/s. In general, RAID Level 1 needs nearly twice as many disks as RAID Level 5 and has much higher cost. Having more disks, RAID Level 1 generally yields higher performance than RAID Level 5. It then becomes the reader's responsibility to synthesize this performance and cost data.

A second method to combine performance and cost is to divide the performance by the number of disks. As this only makes sense for throughput, response time will be addressed separately. RAID Level 1 has twice as many disks as RAID Level 0, and so we divide RAID Level 1 throughput by two to normalize relative to RAID Level 0. By dividing the throughput by the number of disks, we are tacitly assuming that a RAID Level 0 with *2D* disks should perform twice as well as a RAID Level 0 with *D* disks. This assumption can be *false* if performance is not *disk limited*. In general, we may be unfairly penalizing RAID Level 1 because we are not providing RAID Level 1 with twice the total resources (processor, memory, etc.) of RAID Level 0. We are only doubling the disk resources. In particular, if CPU power is the limiting factor to performance, then doubling the disk resources will not double throughput. CPU power tends to limit performance as more disks are used. Therefore, we limit the number of disks used to ensure that CPU power does not greatly impact performance and RAID Levels with more disks are not unfairly penalized. We will limit almost all experiments in this paper to 20 disks or less. To check that this method of presenting data is fair, we verify that throughput per disk remains constant as we scale the number of disks. This second method of combining throughput and cost is the one used in this paper.

Although throughput scales with the number of disks used, response time does not. We define the 90th percentile response time to be the time in which 90% of the requests in the run were serviced, similar to [Anon 85]. For example, a 90th percentile response time of 1 second would mean that 90% of all requests in that run returned to the user within 1 second. To maintain a valid comparison between RAID Levels, we vary the rate of requests to force different RAID Levels to have the same 90th percentile response time. With this equal 90th percentile response time, we then compare the throughput of the different RAID Levels.

In summary, we compare different RAID Levels by:

(1) maintaining equal user data capacity to simplify the equalization of workloads

(2) forcing comparable 90th percentile response time for all RAID Levels

(3) measuring throughput and dividing by the number of disks involved to get throughput per disk as the main performance metric.

6. Experiment Implementation

Experiments were run on an Amdahl mainframe under UTS, which is a version of System V Unix. See Table 1 for hardware characteristics. To prevent channel conflicts, we use only one disk per string. This approximates a system which uses buffers to avoid data transfer conflicts, as was assumed in [Patterson 88].

By using synthetic workloads instead of application software, we eliminate the need for a file system structure. Instead, we simply read and write bytes on the disks. This enables us to simulate a large range of I/O access patterns without dealing with the logistics of many benchmarks. In our experiment, the reads and writes are done by user processes accessing raw devices [UTS 88].

To achieve a certain target 90th percentile response time, we control the number of outstanding logical requests in the system (*queue depth*). We periodically check the number of requests that were satisfied within the target 90th percentile response time, and adjust the allowed number of outstanding logical requests accordingly.

The parameters of the synthetic workload are 90th percentile response time target, read/write ratio, request size distribution, and data distribution. When a request is generated, we stochastically choose read or write, request size, and starting location of the data. Each choice is made independently of past choices. Note that these parameters determine the logical request stream and are independent of the RAID organization (the logical to physical mapping).

The request size is generated in a number of different ways, depending on the workload. One method is to force all accesses for a run to be a fixed size. This is the assumption used in the simple model, for example, 100% full stripe requests. Since these experiments are run with 10 data disks and track striping, a full stripe is 10 tracks (400 KB) and an individual request is 1 track (40 KB). A second method for choosing request sizes is to use a distribution. We choose two distributions in particular: an exponential distribution with a small mean, approximating transaction processing and UNIX file systems, and a normal distribution with a large mean and standard deviation, approximating supercomputing and image processing workloads.

Because logical requests are logically contiguous and striped over all disks, a logical request's location is specified by the location of its first block: a disk number and a stripe number. We select the stripe number of each request uniformly among all stripes to ensure random seeks.

For most of this paper, we choose the starting disk according to the *aligned* distribution. This is intended to be the data distribution which yields optimal performance. The aligned data distribution tries to 1) minimize the number of partial stripes, and 2) spread the I/O load evenly across disks. It minimizes the number of partial stripes by aligning data on full stripe boundaries where possible. For request sizes smaller than a full stripe, it chooses the starting disk according to a uniform distribution. In the second method, the *unaligned* data distribution, we no longer make any attempt align data on full stripe boundaries. We also create hot disks by favoring certain disks.

7. The RAID Paper Workload

The simple model analyzed performance under four types of workloads: large reads, large writes, small reads, and small writes. While continuing to direct our experiments toward understanding these four types of workloads, we seek to make them more variable as discussed in Section 4. Rather than jump from the analysis presented in the RAID paper directly to our most variable workload, we make the transition in a number of smaller steps.

Request sizes in the RAID paper workload are large (a full stripe) or small (individual blocks). Runs are either 100% reads or 100% writes. With the RAID paper assumption of infinite workload, disks are always 100% utilized, and data distribution has no effect. To approximate this situation, we use a very high workload and the aligned data distribution.

7.1. Idle System Request Latency

To understand the hardware of this experiment, we first break down the time of a basic I/O. This is done by analyzing the average response time of a single request in an idle system. We

Processor Resources: Amdahl 5890-300e	
processors	2
cycle time	15 ns
memory size	256 MB
data cache	64 KB
instruction cache	32 KB
channels	64

Disk Resources: Amdahl 6380	
cylinders/disk	885
tracks/cylinder	15
sectors/track	10
bytes/sector (fixed format)	4 KB
average seek	15 ms
average rotational latency	8.3 ms
maximum transfer rate	2.4 MB/s
disk caching	off
disk scheduling	FIFO

Table 1: Hardware Statistics of Processor and Disk Resources [Amdahl 5890, Amdahl 6380].

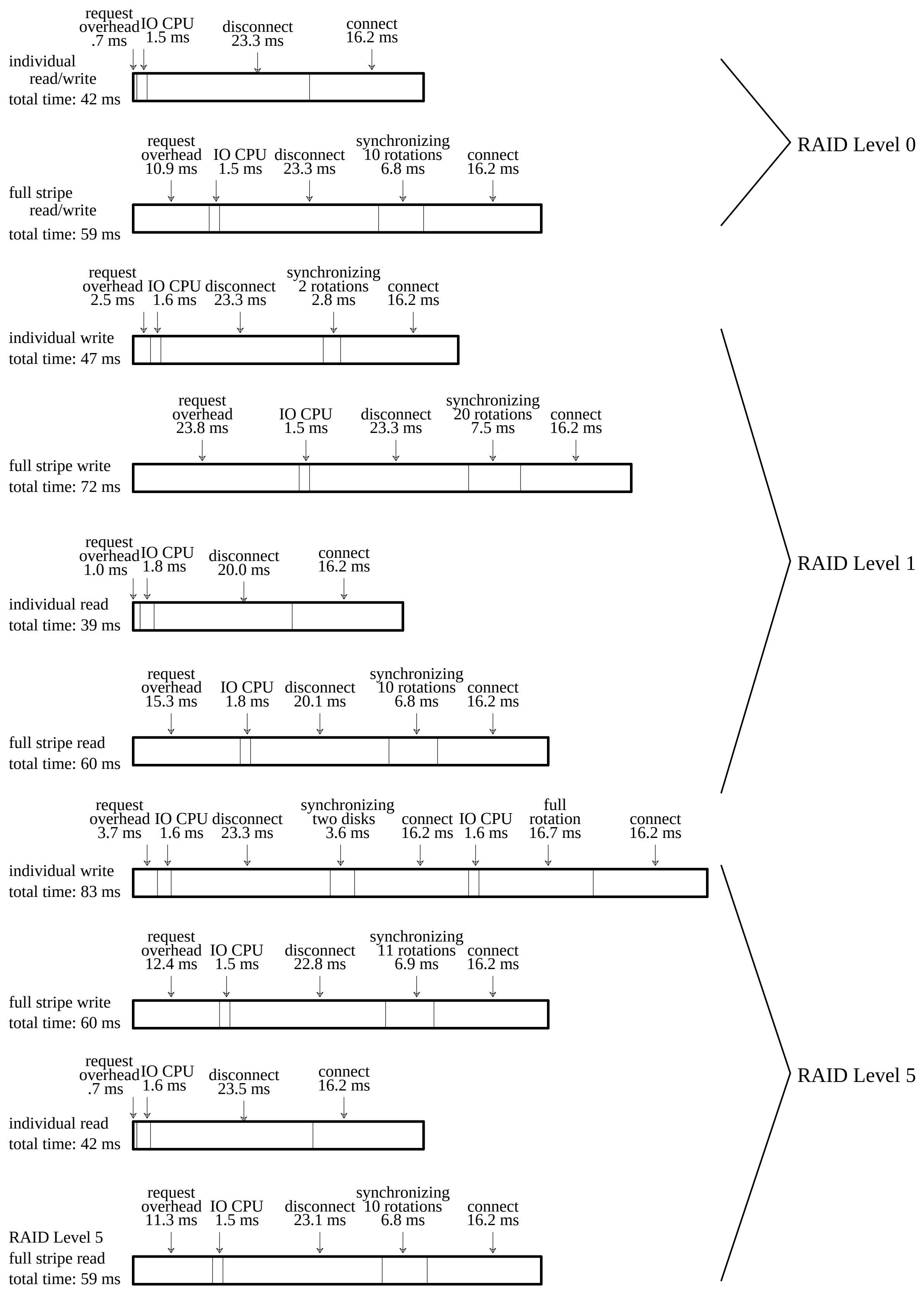


Figure 3: Lifetime of Requests. The lifetime of each type of request is traced by measuring the time spent in various stages of servicing the request: request overhead, IO CPU time, disconnect, synchronization, and connect.

trace the lifetime of an average request by measuring and modeling the time spent in various stages of servicing the request (Figure 3). We also measure the average end-to-end response time of a request and check that this is equal to the sum of the average times spent in each stage. In all cases, the average response time was within 2% of the sum of the average time spent in each stage.

Average response time breaks down as follows:

- *request overhead*: time spent by UTS in issuing a logical request.
- *IO CPU time*: measured CPU time for the device driver to issue and receive I/O's, including the channel processing time. IO CPU time ranges from 1.5 ms to 1.8 ms.
- *disconnect time*: measured time spent in seek and rotational latency.
- *synchronization*: additional time due to multiple independent disks doing random seeks and rotations. This is not measured, but rather calculated based on statistics given in Section 6. Also see Section 7.1.1.
- *connect time*: measured time spent in transferring data [IBM 87]. For full track data transfers, connect time is 16.2 ms.

Note that request overhead is measured *per logical request*. IO CPU time, disconnect time, and connect time are all measured *per disk access*.

7.1.1. Synchronizing Multiple Disks

There are two types of synchronization between multiple disks, seek distance and rotational latency. In most cases, such as RAID Level 0 full stripe reads or writes and RAID Level 1 individual writes, only rotations need to be synchronized. Seek distance among disks that cooperate in a request are usually the same—the sole exception being RAID Level 5 small writes. Because workloads are homogeneous, disks that cooperate in any request cooperate for all requests. For example, for RAID Level 1 individual writes, each data or shadow disk pair always seek to the same track. Thus, for RAID Level 0 full stripe reads or writes, RAID Level 1 full stripe reads or writes and individual writes, and RAID Level 5 full stripe reads or writes, multiple rotations lengthen the total request time in proportion to the number of disks involved. Synchronizing N multiple rotations is equivalent to taking the maximum of N uniform random variables distributed between 0 and 16.7 ms. The expected value of such a distribution is $\frac{N}{N+1} \times 16.7$ ms. The difference between this expected value and the average rotational latency of one disk (8.3 ms) is the penalty for synchronizing multiple rotations.

The second type of synchronization, synchronizing both seeks and rotations, is relevant only for RAID Level 5 individual writes. The RAID paper assumed that a RAID Level 5 individual write was four equal disk accesses. Because the four disk accesses are issued to two disks, each disk sees a pair of requests: first a read of the current data or parity, then a write of the new data or parity. The lifetime shown in Figure 3 focuses on one of these disks, say the data disk. Note the two distinct disk accesses in the broken out trace of RAID Level 5 individual writes in Figure 3. For the first disk access (the read), we need to add synchronization because we are using two independent disks. However, because parity is on different disks for different stripes, two disks that cooperate in one request do not necessarily cooperate in the next request. Thus, two disks in a request could have different current head positions. Because of this, we need to synchronize not only different rotations, but also different seek distances. Just as we synchronized multiple rotations by taking their maximum, we synchronize multiple *seek + rotations* by taking the maximum of a number of random variables, each distributed as a *seek + rotation*. By Monte Carlo simulation, we find this adds approximately 3.6 ms to an average seek plus an average rotation.

For the second disk access (the write), no seek is needed. Because the disk has just finished reading the old data, the new data can be written without moving the disk head. However, we do see a full rotation instead of an average rotation. Thus, we save an average seek (15 ms) but pay an extra half rotation (8.3 ms). We refer to this as RAID Level 5 *saving a seek*.

7.1.2. Request Overhead

Request overhead changes drastically with the number of disks involved in a request. For example, for RAID Level 0, request overhead jumps from .7 ms for individual reads/writes to 10.9 ms for full stripe reads/writes. This drastic increase is a side effect of running on an idle system. In an idle system, queues at the device driver are empty, and each disk access of a multiple disk request starts as soon as that access reaches the driver queue. Starting these disk accesses delays the rest of the logical request from being issued. Under normal loads, disks are usually working on a previous request, and the entire logical request is free to be issued without waiting for any disk I/O's to start. For example, for full stripe requests to a non-idle system, the request overhead is approximately 3.5 ms per logical request.

Request overhead also increases with the number of total disks in the system. This is due to the increased overhead in managing more disks. In general, this effect is much less pronounced on a non-idle system. However, it is true that RAID Level 1's 20 disks require more CPU power than RAID Level 0's 10 disks. By limiting the experiments to 20 disks as discussed in Section 5.2, we prevent this increased CPU demand from unfairly penalizing the throughput per disk of RAID Level 1.

7.1.3. Seek Optimization

RAID Level 1 reads are almost identical to RAID Level 0 reads. A key difference, however, is the disconnect time. The disconnect time of RAID Level 1 is shorter because RAID Level 1 requests can choose between two disks which have the same data. By choosing the copy of the data which results in the shorter seek time, the average disconnect time drops 3-4 ms from RAID Level 0 [Bitton 88].

7.2. Analyzing a Saturated System

As discussed in Section 6, we control system load by controlling the number of outstanding logical requests in the system. Increasing the target 90th percentile response time allows the system to have more outstanding logical requests. This causes higher disk utilization and correspondingly higher throughput. By varying the load, we can graph absolute throughput per disk versus 90th percentile response time. Figures 4 show these graphs for the RAID paper workloads: large reads, large writes, small reads, and small writes. As before, large means a full stripe; small means an individual request (one track).

Note that the minimum 90th percentile response times are those discussed in Section 7.1. As expected, when we allow requests to have longer service times, we see higher throughput. Eventually, when the disks are fully utilized, increasing the 90th percentile response time no longer increases the throughput. The RAID paper analysis deals with this point of *maximum throughput*. By presenting RAID Levels 1 and 5's maximum throughput per disk relative to RAID Level 0's, we can make a direct comparison to the performance predicted in the RAID paper (Figure 5).

Figure 5 shows that, for most of these workloads, the simple model in the RAID paper accurately predicts the actual performance of real hardware. However, there are significant differences. In RAID Level 1, relative throughput for reads is higher than predicted by our simple model. Recall that the simple model assumed all disk accesses took a constant amount of time.

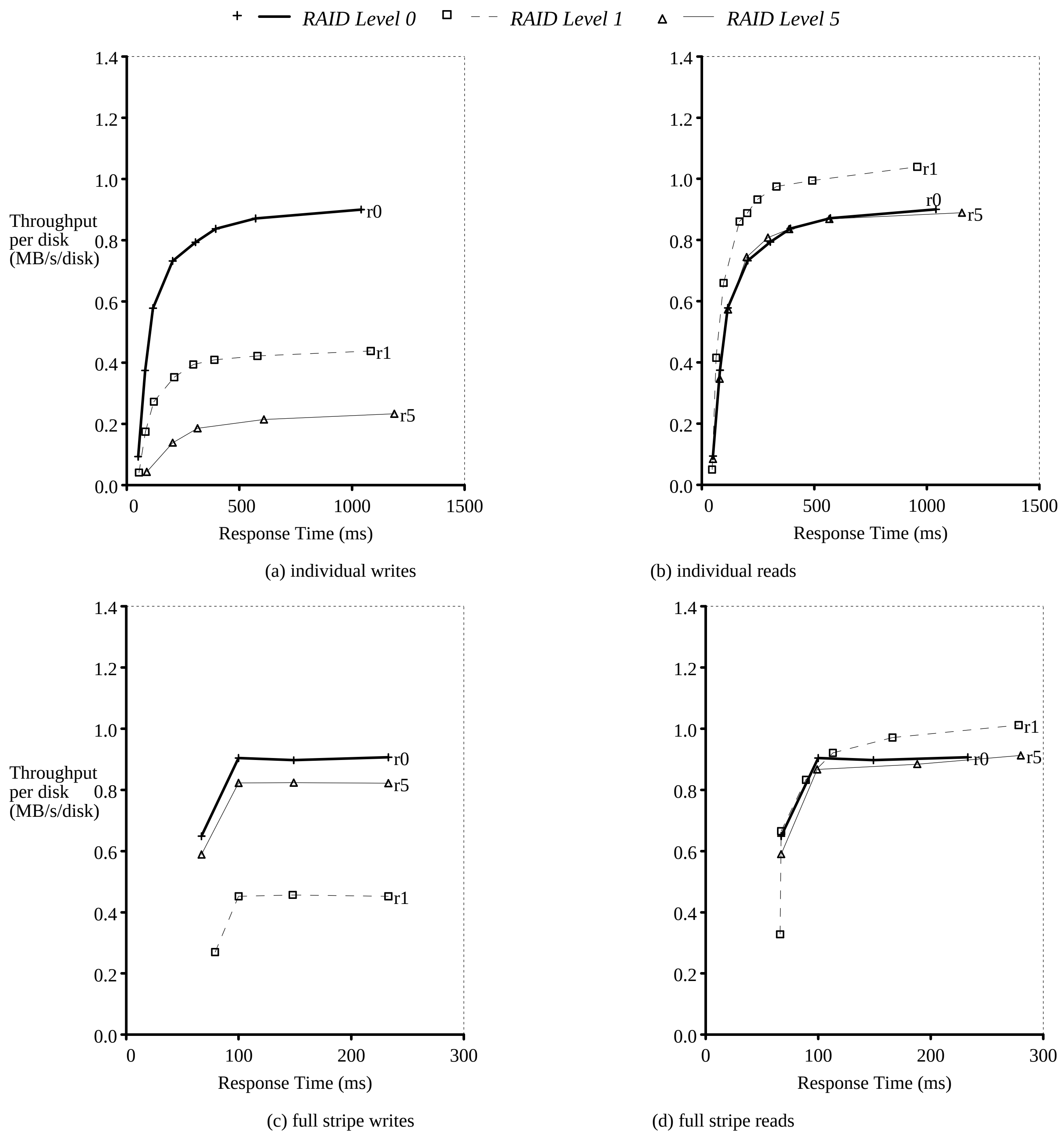


Figure 4: Throughput vs. Response Time. Throughput is graphed as a function of 90th percentile response time target for 4 types of I/O requests: individual reads and writes, full stripe reads and writes. These graphs were generated by keeping a fixed number of logical requests in the queue and measuring the resulting throughput and 90th percentile response time. The data distribution used here is aligned. Because of the random distribution of requests, workloads with individual requests require a much higher load to saturate the disk system than workloads with full stripe requests.

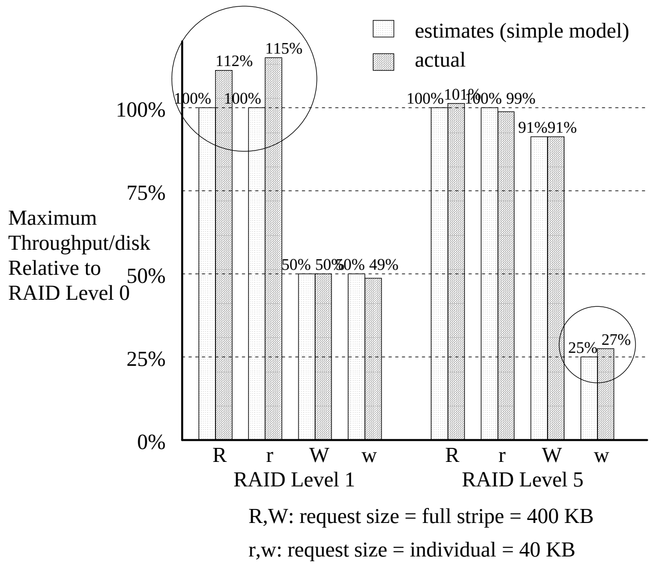


Figure 5: Comparing Maximum Throughput Against Simple Model. The measured maximum throughput per disk relative to RAID Level 0 is compared against the estimates made in the RAID paper [Patterson 88]. The only significant differences occur for RAID Level 1 reads, due to seek optimization, and RAID Level 5 individual writes, due to saving a seek.

Because of seek optimization, this assumption is no longer valid. We adjust the simple model by defining relative throughput as the total disk-time used by a RAID Level 0 logical request divided by the total disk-time used by the RAID Level in question. When seen in this light, we can easily adjust for different access times. Due to seek optimization, the disk-time for RAID Level 1 reads is 4 ms per disk less than RAID Level 0. This represents approximately 10% of the total disk-time, so the adjusted simple model predicts a relative throughput of 110% for RAID Level 1. The measured relative throughput is close to this prediction, ranging from 112% - 115%.

A RAID Level 5 small write causes two disks to perform a seek, an average rotation, a full track transfer, a full rotation, and a second full track transfer (each disk takes 72 ms, yielding a disk-time of 144 disk-ms). The total disk-time for RAID Level 0 small writes is approximately 40 disk-ms. Thus, while the simple model predicts a relative throughput of 25%, the adjusted simple model predicts a relative throughput of 40/144 = 28%. This agrees with the measured performance. The adjusted model does not change the relative throughput estimates for other RAID Levels and workloads.

7.3. Maintaining Equal Response Times

Section 7.2 compared the maximum throughputs of different RAID Levels. However, the different RAID Levels reach maximum throughput at different response times. It is unfair to compare RAID Levels at different target response times. By forcing all RAID Levels to have equal 90th percentile response times, we generate a fairer comparison. Figure 4 showed that throughput for all RAID Levels drops as the 90th percentile response time target decreases. To better understand how throughput changes with response times for each RAID Level, we graph throughput per disk versus 90th percentile response time in a slightly different way. Instead of absolute throughput per disk, we graph the *percentage of each RAID Level's maximum throughput*. For example, since RAID Level 0's maximum throughput per disk is .906 MB/s/disk for small writes, RAID Level 0 throughput is graphed as a percentage of .906 MB/s/disk in Figure 6a. RAID Level 1's maximum throughput per disk for small writes is .438 MB/s, so throughput per disk is graphed as a percentage of .438 MB/s in Figure 6a.

These response time behavior graphs show us what to expect when we maintain equal 90th percentile response times for each RAID Level. For example, in Figure 6a, we see that at any 90th percentile response time less than 1000 ms, RAID Level 5 achieves a lower percentage of its maximum throughput than RAID Level 0. This is caused by RAID Level 5 small writes' longer idle response times (Section 7.1.1). Thus, RAID Level 5's relative throughput (relative to RAID Level 0) for small writes will decrease. In contrast, RAID Level 1 small writes follow RAID Level 0 small writes very closely at all 90th percentile response times. Thus we expect RAID Level 1 small writes to maintain the same relative throughput.

In the remainder of this paper, we will use one specific 90th percentile response time for each workload. The target 90th percentile response time for each workload is set at four times the minimum 90th percentile response time for that workload on an idle RAID Level 0 (Section 7.1). That is, if 90% of all requests on an idle RAID Level 0 return in time *t*, we control the response time such that 90% of all requests return in time *4*t*. For full stripe requests, target 90th percentile response time is 268 ms; for individual requests, target 90th percentile response time is 200 ms[1]. This is intended to allow some freedom to queue requests in order to achieve higher throughput without allowing response times to grow too large. Notice that four times the idle RAID Level 0 90th percentile response time for large requests easily exceeds the 90th percentile response time needed to achieve maximum throughput. An interesting future experiment would be to restrict large requests to a small percent over the minimum 90th percentile response time.

Figure 7 shows the relative throughput per disk of RAID Levels 1 and 5 before and after equalizing to these target 90th percentile response times. Note that at the particular 90th percentile response times we chose, only two workloads show significant change relative to RAID Level 0: RAID Level 5 small writes and RAID Level 1 small reads. We discussed previously how the throughput for RAID Level 5 small writes changed as we maintained equal 90th percentile response times. Similarly, Figure 6b shows that, at our 200 ms target 90th percentile response time, RAID Level 1 is at a slightly higher percentage of its maximum throughput than RAID Level 0.

This section has focused on 90th percentile response times. We have seen that limiting 90th percentile response times causes less than maximum throughput for all RAID Levels. In general, RAID Levels with longer idle response times are penalized more than RAID Levels with shorter idle response times.

8. More Variable Workloads

In Section 7, we analyzed the RAID paper workload of full stripe or individual requests, 100% reads or 100% writes. We started by analyzing idle systems (minimum 90th percentile response time) and saturated systems (maximum throughput). We finished by analyzing systems with equivalent 90th percentile response times. While still maintaining equivalent 90th percentile response times, we now begin modifying the workload components: request size, data distribution, and read/write ratio.

[1]The average response times at these target 90th percentile response times are 230 ms for full stripe requests and 104 ms for individual requests.

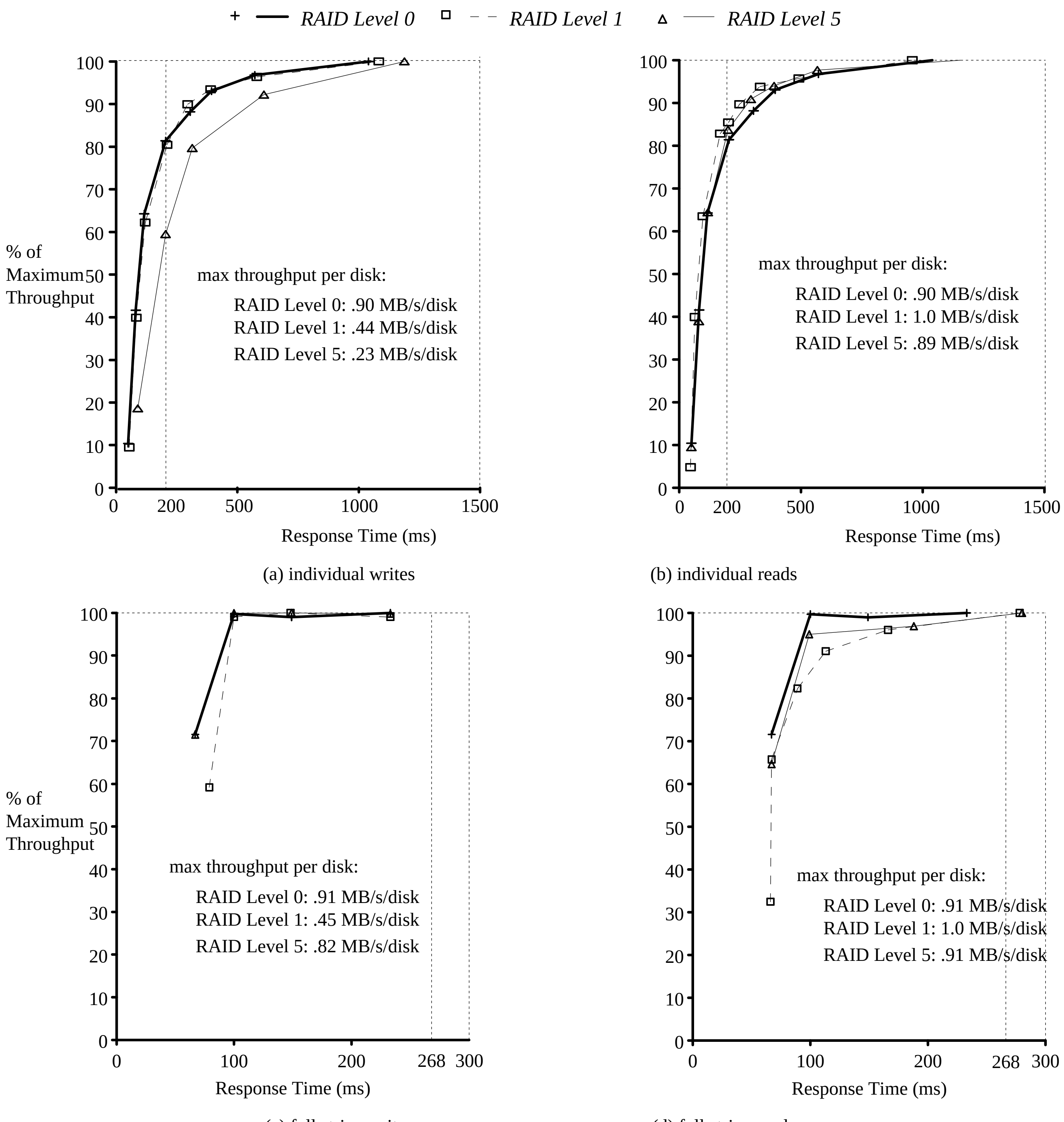


Figure 6: Percentage of Maximum Throughput. Throughput per disk for each RAID Level is shown as a percentage of the maximum throughput per disk for that RAID Level. Thus, we can see the relative effects of changing the 90th percentile response time target. For example, in Figure 6a, RAID Level 5 achieves a lower percentage of its maximum throughput than either RAID Level 0 or 1 at all target response times. Individual reads and writes are 40 KB; full stripe reads and writes are 400 KB. Data distribution is aligned.

8.1. Distributing Request Sizes

A key characteristic of any workload is the logical request size distribution. Until now, all large requests have been full stripe requests and all small requests have been individual requests. Thus, with 10 disks and track-level striping, large requests have been exactly 400 KB and individual requests have been exactly 40 KB. However, in real-world systems, large requests are often much larger than 400 KB [Bucher 80] and small requests are often much smaller than 40 KB [Ousterhout 85, Anon 85]. Also, applications rarely issue requests that are all the same size. We therefore make two changes to the request size distribution:

(1) We no longer restrict the workloads to one particular size. Rather, we use a *distribution* of request sizes. For large requests, we generate request sizes derived from a normal distribution; for small requests, we generate request sizes based on an exponential distribution.

(2) We change the average size of both large and small requests: an average large request changes from 400 KB to 1.5 MB (approximately 4 stripes); an average small request changes from 40 KB to 6 KB, the closest we could come to one 4KB sector.

Thus, the large requests get larger and the small requests get smaller. This significant change will alter many of our results; qualitatively, however, what we have learned so far will still hold. Seek optimization will continue to benefit RAID Level 1 reads; RAID Level 5 small writes will still save a seek. Changes to our target 90th percentile response times will follow the changes in RAID Level 0 minimum 90th percentile response times (new 90th percentile response time targets will be 780 ms for large requests and 148 ms for small requests).

Because large requests using our new size distributions will usually cover multiple stripes, each disk transfers more information per seek than before and absolute throughput will increase by 60%-80% for all RAID Levels. In contrast, because small requests are smaller on average, less data is transferred per seek and absolute throughput for small requests will decrease 75%-80% for all RAID Levels. In Figure 8, we show the relative throughput before and after we distribute and change the means of the request sizes. The following four subsections discuss these results.

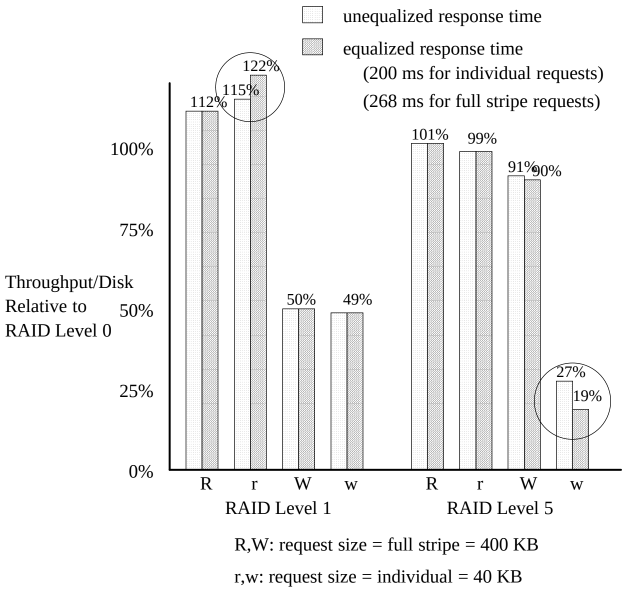


Figure 7: Before and After Equalizing Response Times. Throughput per disk relative to RAID Level 0 is shown before and after maintaining equal 90th percentile response times. We see that equalizing to our target 90th percentile response times affects relative throughput for RAID Level 1 individual reads and RAID Level 5 individual writes.

8.1.1. RAID Level 1

With larger requests, each disk access takes longer. Because of this, savings due to seek optimization are a smaller fraction of the access time of each disk. Thus, RAID Level 1 large reads, which benefit from seek optimization, show less performance advantage over RAID Level 0. However, we do expect RAID Level 1 to still have slightly higher throughput per disk than RAID Level 0. Unfortunately, the request overhead for RAID Level 1 is 3-4 ms higher than for RAID Level 0. This cancels out the slight performance advantage of seek optimization, and makes RAID Level 1 throughput for large reads equal to RAID Level 0.

With the other workloads, large writes, small reads, and small writes, the relative throughput of RAID Level 1 remains unchanged.

8.1.2. RAID Level 5 Small Writes

As shown in Figure 8, RAID Level 5 small writes improve in relative throughput per disk. Two factors combine to cause this improvement. First, by saving a seek on the data and parity write, we save a fixed amount of time. With smaller requests, the total request time is shorter and this seek savings is a slightly larger fraction of the entire request time. This larger seek savings accounts for 1% of the 6% change we see in Figure 8.

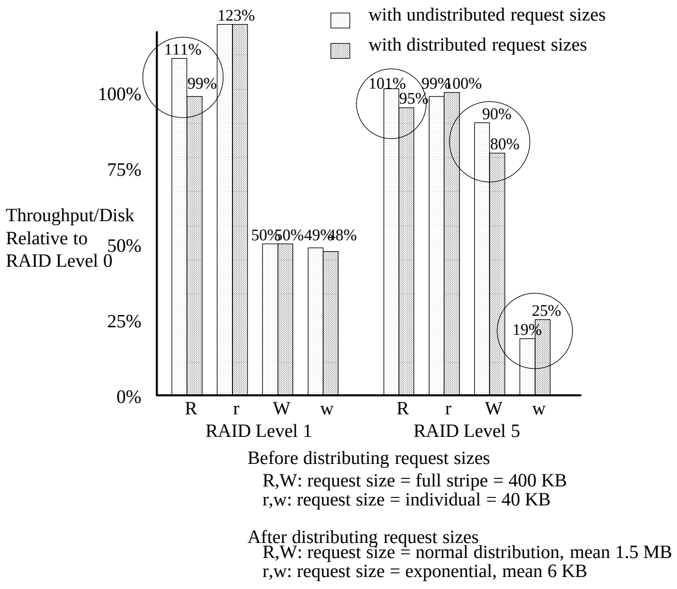


Figure 8: Relative Throughput Before and After Changing Request Sizes. In this figure, we see how the throughput of RAID Levels 1 and 5 change relative to RAID Level 0 as request sizes are distributed. Distributing the request sizes decreased the relative throughput per disk for RAID Level 1 large reads due to the lessening importance of seek optimization. RAID Level 5 large reads decreased in relative throughput due to reading the parity tracks. RAID Level 5 large writes decreased in relative throughput due to the presence of partial stripe writes. RAID Level 5 small writes increased in relative throughput due to a different 90th percentile response time target. Data distribution is aligned.

Most of the improved relative throughput is due to selecting a 90th percentile response time target which is more favorable for RAID Level 5 small writes than the 90th percentile response time target for undistributed request sizes. In Figure 6a, throughput at the target response time (200 ms) was at 59% of maximum throughput for RAID Level 5 and 81% for RAID Level 0. After distributing requests sizes, a graph similar to Figure 6a would show that throughput at the target response time (148 ms) remains roughly the same for RAID Level 0 (85%). However, this graph would also show that RAID Level 5 at this response time is closer to its maximum throughput (72%) than before. As a result, RAID Level 5's relative throughput increases.

8.1.3. RAID Level 5 Large Reads

For full stripe reads (before distributing request sizes), the relative throughput of RAID Level 5 was 100%. This result was expected, as no extra redundant information needed to be read, and the number of disk accesses in the physical request equaled the number of disk accesses in the logical request. However, with larger requests, covering multiple stripes, we can no longer simply read the data and ignore parity. For example, in Figure 1c, if the logical request reads logical tracks 0-13, disk 1 must read its physical tracks 0, 2, and 3. Because the experiment is run in user-level, we do not have control of the channel program. Rather, because physical tracks 0, 2, and 3 are not contiguous, we must issue two separate I/O's to disk 1. First, read physical track 0. Second, read physical track 2-3. Unfortunately, by the time the driver is able to complete the first I/O and issue the second I/O, the disk has rotated enough to miss the start of track 2. Thus, issuing two I/O's in order to skip over the parity track costs a full rotation, the same as simply reading the parity track and issuing one large I/O.

An average size (4 full stripes, or 40 tracks) request will need to skip over two parity tracks. The overhead, then, is 2/40 or 5%. This accounts for the 5% drop in throughput shown in Figure 8.

8.1.4. RAID Level 5 Large Writes

For writes which exactly cover a number of full stripes, performance is straightforward. To maintain the parity information, one parity track must be written for every 10 data tracks. Full stripe writes do not need to read any information. However, when request sizes become distributed, most requests will not cover an exact number of full stripes. Usually, one or both ends of the request will be a partial stripe. For example, in Figure 1c, if a logical request covers logical tracks 0-13, stripes 0, 1, and 2 are full stripe writes but stripe 3 is only *partially* written (logical tracks 12 and 13). Thus, although stripes 0-2 act as full stripe writes with relative throughput $\frac{N-1}{N}$, stripe 3 acts as a partial stripe write. As discussed in Section 3, partial stripe writes can have relative performance ranging from 25% to $\frac{N-2}{N}$, depending on the size of the partial stripe.

In this section, we allow at most one partial stripe per request. This is done by using the *aligned* data distribution (see Section 6). With this data distribution, requests larger than one full stripe are forced to either begin or end at a full stripe boundary. Introducing one partial stripe per request causes the relative throughput of RAID Level 5 large writes to drop from 90% to 80%. When we use data distributions that do not align data on full stripe boundaries, we see up to two partial stripes per request, and another 10% drop in relative throughput of RAID Level 5 large writes. As requests get larger and partial stripes become a smaller fraction of the entire request, this penalty for partial stripe writes will decrease. For instance, for requests over 100 MB, the penalty for partial stripe writes might be less than 1%. In the remainder of the paper, we use the unaligned data distribution and allow more than one partial stripe per request.

8.2. Mixing Reads and Writes

Many different types of workloads exist in the real world. Very few, if any, are 100% reads or writes. We have used 100% reads or writes to better understand how varying other workload parameters affects performance. Now, keeping equalized 90th percentile response times, distributed request sizes, and unaligned data distributions, we at last mix reads and writes. In Figure 9a, we see that, for large requests, at almost all mixtures of reads and writes, RAID Level 5 has significantly higher throughput per disk than RAID Level 1. For small requests (Figure 9b), we see that, because of RAID Level 5's poor small write performance and RAID Level 1's seek optimization for reads, RAID Level 1 has higher throughput per disk than RAID Level 5. Notice that throughput at all intermediate read/write ratios is not well-modeled by a weighted mixture of the 100% read and 100% write performance. Such a linear model is generally optimistic.

9. Summary

We have started with a simple model for performance and, step by step, measured performance with more and more variable workloads. We first measured maximum throughput using the RAID paper workload of 100% reads or writes, individual or full stripe requests. We found that the simple model accurately predicted performance for most cases. The main exception was RAID Level 1 reads, where seek optimization causes higher than expected throughput.

Second, we equalized 90th percentile response times. We found that equalizing response times hurt the relative throughput of RAID Levels with higher idle system response times (RAID Level 5 small writes) and helped the relative throughput of RAID Levels with lower idle system response times (RAID Level 1 small reads).

Third, we distributed the request sizes and made large requests larger and small requests smaller. We found that seek optimization ceased to noticeably help RAID Level 1 large reads. RAID Level 5 large reads were penalized for skipping parity tracks. RAID Level 5 large writes suffered from partial stripe writes. Even more partial stripe writes were generated by allowing unaligned requests.

In conclusion, we look at Figure 9. Note that for small requests, RAID Level 1 consistently yields higher throughput per disk than RAID 5. In contrast, for large requests, RAID Level 5 almost always performs better than RAID Level 1. From 0% reads to almost 95% reads, RAID Level 5 yields higher throughput per disk than RAID Level 1. We conclude that for applications with predominantly large requests, RAID Level 5, rotated parity, clearly outperforms RAID Level 1, mirroring.

The full version of this paper appears in [Chen 89] and explores additional issues, such as further varying the request size distribution, varying the unit of interleaving, scaling the number of disks, and connecting more than one disk per string. We are continuing to analyze the performance of disk arrays. In particular, we are interested in arrays of small disks. We have built and are currently evaluating a disk array of 30-50 CDC Wren 5.25" disk drives. One step in that evaluation will be to carry out experiments similar to the ones in this paper. Further work will entail building a file system and running real world benchmarks on that system.

10. Acknowledgements

These experiments were made possible by the generous support of Amdahl Corporation, which provided many hours of

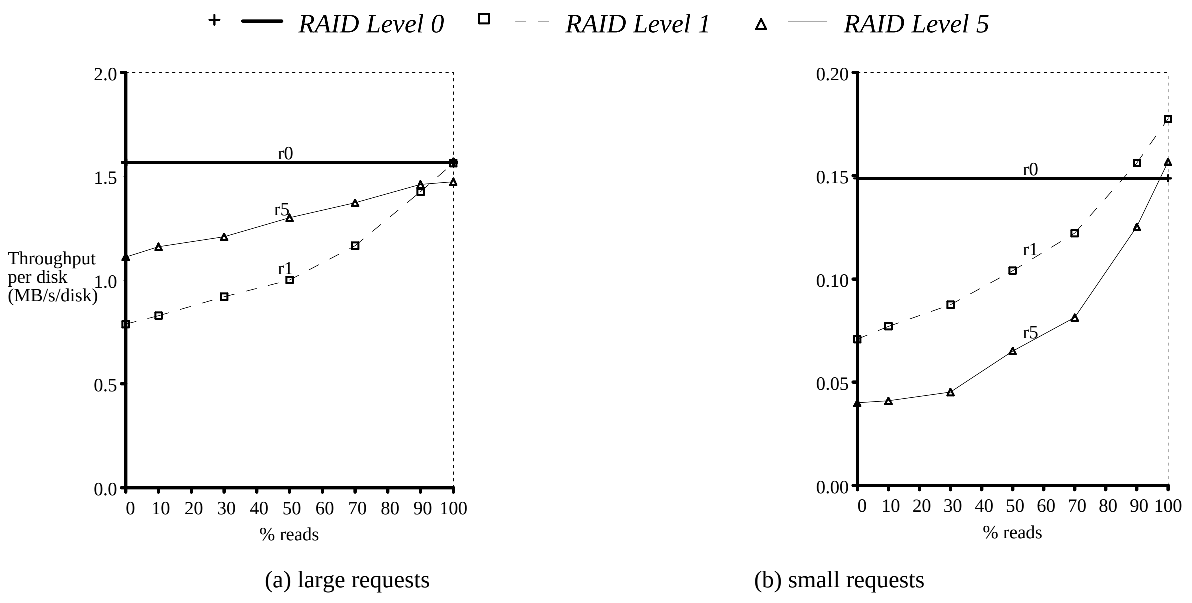


Figure 9: Effect of Mixing Reads and Writes. Throughput per disk is graphed as a function of the percentage of reads in the workload. RAID Level 0 reads and writes are assumed to be equivalent, and are shown as a horizontal line in both graphs. The size distribution for large requests is a normal with a mean of 1.5 MB (90th percentile response time target of 780 ms). The size distribution for small requests is an exponential with a mean of 6 KB (90th percentile response time target of 148 ms). Data distribution is unaligned.

expensive standalone machine time. John Colgrove, Art Casabal, and Romeo Narciso provided much needed technical assistance at Amdahl. Susan Eggers, Alan Smith, and Dick Wilmot offered helpful suggestions throughout. Peter Chen is supported by an ONR fellowship, and Garth Gibson is supported by an IBM fellowship. The RAID project is funded in part by the National Science Foundation under grant #MIP-8715235.

11. References

[Amdahl 5890] *‘‘5890 Processors’’*, Amdahl marketing publications MM001388, 1988.

[Amdahl 6380] *‘‘6380E Direct Access Storage Device’’*, Amdahl marketing publications MM001352, 1987.

[Amdahl 67] G. Amdahl, *‘‘Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities,’’* Proceedings AFIPS 1967 Spring Joint Computer Conference, Vol. 30 (Atlantic City, NJ, April 1967), pp. 483-485.

[Anon 85] Anon Et Al, *‘‘A Measure of Transaction Processing Power,’’* Tandem Technical Report TR85.2, 1985.

[Bell 84] C.G. Bell, *‘‘The Mini and Micro Industries,’’* IEEE Computer, Vol. 17, No. 10 (October 1984), pp. 14-30.

[Bitton 88] D. Bitton and J. Gray, *‘‘Disk Shadowing,’’* Very Large Database Conference, 1988.

[Bouknight 72] W. Bouknight, S. Denenberg, D. McIntyre, J. Randall, A. Sameh, D. Slotnick, *‘‘The Illiac IV System,’’* Proc. IEEE, April 1972, pp. 369-388.

[Bucher 80] I. Bucher, A. Hayes, *‘‘I/O Performance Measurement on CRAY-1 and CDC 7600 Computers,’’* 16th Meeting of Computer Performance Evaluation Users Group, Oct. 1980.

[Buzen 86] J. Buzen, A. Shum, *‘‘I/O Architecture in MVS/370 and MVS/XA’’*, CMG Transactions, vol 54, Fall 1986, pp. 19-26.

[Chen 89] P. Chen, *‘‘An Evaluation of Redundant Arrays of Disks Using an Amdahl 5890 (full version),’’* Master’s Thesis, University of California at Berkeley Report No. UCB/Computer Science Division 89/506, May 1989.

[Harker 81] J. Harker et al., *‘‘A Quarter Century of Disk File Innovation,’’* IBM Journal of Research and Development, Vol 25, No. 5, Sept. 1981, pp. 677-689.

[IBM 87] *IBM System/370XA Principle of Operations*, IBM publication SA22-7085-01, 2nd ed. Jan. 1987.

[Joy 85] B. Joy, presentation at ISSCC 1985 panel session, Feb. 1985.

[Kim 86] M. Kim, *‘‘Synchronized Disk Interleaving,’’* IEEE Trans. on Computers, Vol. C-35, no. 11, Nov. 1986.

[Kim 87] M. Kim, A. Nigam, G. Paul, *‘‘Disk Interleaving and Very Large Fast Fourier Transforms,’’* International Journal of Supercomputer Applications, Vol 1, No. 3, Fall 1987, pp. 75-96.

[Moore 75] G. Moore, *‘‘Progress in Digital Integrated Electronics,’’* Proc. IEEE Digital Integrated Electronic Device Meeting, 1975, p. 11.

[Myers 86] W. Myers, *"The Competitiveness of the United States Disk Industry*, IEEE Computer, Vol. 19, No. 11, January 1986, pp. 85-90.

[Ng 88] S. Ng, D. Lang, R. Selinger, "Trade-offs Between Devices and Paths in Achieving Disk Interleaving", The 15th Annual International Symposium on Computer Architecture (SIGARCH 88).

[Ousterhout 85] 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,’’* ACM Operating Systems Review, Vol. 19, No. 5, Proceedings of the 10th ACM Symposium on Operating System Principles, Dec. 1-4, 1985.

[Patterson 88] D. Patterson, G. Gibson, and R. Katz, *‘‘A Case for Redundant Arrays of Inexpensive Disks (RAID),’’* ACM SIGMOD conference proceedings, Chicago, IL., June 1-3, 1988, pp. 109-116.

[Salem 86] K. Salem, H. Garcia-Molina, *‘‘Disk Striping,’’* IEEE 1986 Int. Conf on Data Engineering, 1986.

[Schulze 88] M. Schulze, G. Gibson, R. Katz, and D. Patterson, *‘‘How Reliable is a RAID?,’’* Spring COMPCON 89, March 1, 1989, San Francisco, CA.

[Thisquen 88] J. Thisquen, *‘‘Seek Time Measurements’’*, Amdahl Peripheral Products Division Technical Report, May 9, 1988.

[UTS 88] *UTS580 User Reference Manual*, Amdahl publication ML-142292, 1988.