An Evaluation of Redundant Arrays of Disks using an Amdahl 5890

Peter M. Chen

Report No. UCB/CSD 89/506
May 1989

Computer Science Division (EECS)
University of California
Berkeley, California 94720
An Evaluation of Redundant Arrays of Disks using an Amdahl 5890

Peter M. Chen

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720

ABSTRACT

I/O systems are increasingly becoming a major performance limitation to faster computer systems. 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, request size, and the ratio of reads and writes. We also explore tradeoffs in the unit of interleaving and number of disks. We find that for applications with large accesses, such as many supercomputing applications, a rotated parity disk array far outperforms traditional mirroring architecture. In contrast, for applications with many small accesses, such as transaction processing and traditional file systems, mirroring disk arrays outperform rotated parity disk arrays.

Keywords: disk arrays, performance, I/O, RAID
An Evaluation of Redundant Arrays of Disks using an Amdahl 5890

1. The I/O Crisis

Over the past 10 years, processing speed, memory speed and 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]. If not remedied, this imbalanced system growth will eventually lead to I/O limited systems [Amdahl 67, Kim 87]. Continued improvement in system performance depends in a large part on I/O systems with higher data rate and I/O rate.

One way to increase I/O performance is by using an array of many disks [Kurzweil 88]. 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 several disk array redundancy schemes. The performance analysis is based on a set of experiments carried out on Amdahl hardware. In these experiments, we explore several issues:

- What are the basic differences in throughput and response time between the various redundancy schemes?
- For each redundancy scheme, how do different response time requirements affect throughput?
• 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?
• What affect does interleaving data in different units have on performance?
• How do the redundancy schemes scale with increasing numbers of disks?

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 additional RAID Level 0. RAID Level 0 is a non-redundant array of disks, and is added mainly 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 the 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 1b 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.
(a) RAID Level 0—non-redundant array  (b) RAID Level 5—rotated parity. The shaded areas are parity.

(c) RAID Level 1—mirroring. Shadow disks are shaded.

**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. Each stripe of data consists of a logical block from each disk. In RAID Level 1 (Figure 1c), each data disk (disks 0-3) has a shadow disk (disks 4-7). In RAID Level 5 (Figure 1b), parity for each stripe is kept in a parity block. Which physical disk the parity block is kept on is different for different stripes.

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 parity information onto disks. As shown in Figure 1b, 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. The 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, the physical request will involve more disk blocks than the 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 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 the RAID Levels that we are concerned with, 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 the corresponding 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 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 1b, 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 stripe 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 stripe 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. Goals and Refinements

Our overall goal is to understand more fully how RAID Levels 1 and 5 perform. This includes exploring aspects of implementation, workload characterization, and performance evaluation. In
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 ($N=11$).

In particular, the performance estimates above dealt with maximum possible throughput, expressed in terms of relative throughput. A specific goal of the experiments described here is to \textit{measure} the performance of RAID Levels 1 and 5 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:

- \textit{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.
- 7 -

- **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, it used extremely simple, therefore unrealistic, workloads. These experiments refine the workloads in three ways:
  
  1. 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.
  
  2. The RAID paper workloads were either 100% reads or 100% writes. These experiments explore a range of read/write ratios.
  
  3. 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.

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. Also, running application programs require an underlying file system, which we do not have.

5. Comparing RAID Levels

When comparing RAID Levels, we are interested in performance (throughput and response time) and cost. Comparing RAID Levels in these two areas is no easy task. Because the storage efficiency differs between RAID Levels 0 (100%), 1 (50%), and 5 \( \frac{N-1}{N} \), RAID systems with the same user data capacity need different numbers of disks. Alternatively, with a fixed number of disks, different RAID Levels will have different user data capacities. There are at least two ways to address this issue.

5.1. Constant Number of Total Disks

The first option is to keep the total number of disks constant between RAID Levels Comparing costs with this option is trivial. Since all RAID Levels use identical hardware, cost is equal. However, comparing the performance of such RAID systems is tricky. To have a valid basis for comparison, equal work-
loads must be presented to all systems. Unfortunately, it is unclear how to define an equal workload between systems with different 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 the data in B is accessed half as frequently as the data in A. To compensate, we may wish to present B with double the workload that A receives. Unfortunately, it is unclear what constitutes double a workload: double the request rate? Double the request size? Double the unit of interleaving? The second option in comparing RAID Levels takes a different approach to resolve this difficulty.

5.2. Constant Number of Data Disks

The second option, which is the one used in these experiments, maintains equal data capacities between RAID Levels. Presenting identical workloads to each system makes comparing performance far simpler. 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. Then, 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 (see section 10.3). 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 (pro-
cessor, 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 up the number of disks used (see section 10.3). 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. Unless otherwise mentioned, response time in this paper is defined as the time in which 90% of the requests in the run were serviced, similar to [Anon 85]. For example, a 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 and force different RAID Levels to have the same response times. With this equal 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 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.

5.3. Two Entire Systems

As a final note, another approach to comparing RAID Levels is possible. We show this option in Figure 3. This approach is easiest to explain for comparing RAID Level 0 and 1, but it also generalizes to RAID Level 5.

Consider a system with $D$ disks and one CPU. Conceptually, we will form a RAID Level 1 with two such systems, systems A and B. Each system in the RAID Level 1 will contain half the data disks along with their corresponding shadow disks. Requests to the RAID Level 1 as a whole will be serviced in part by each of the two systems. Requests that span more than one disk will be broken up and serviced in part
Figure 3: Alternate Way to Compare RAID Levels. This figure shows a way to map RAID Level 1 onto two distinct systems A and B. Even numbered logical blocks are stored in System A; odd numbered logical blocks are stored in System B. Each system will receive exactly half the total workload. Thus, to estimate total throughput, we need only measure the throughput of one of the systems. This has the advantage of needing only half the disk and CPU resources. Also, the system that we measure will have the same CPU resources per disk as a RAID Level 0.

by system A and in part by system B. Similarly, half the requests that need only one disk will be serviced by system A and half will be serviced by system B. The key to this method is that each system will see an identical load. Because of this, we need only measure the throughput of one of these systems and multiply by two to calculate the throughput of the entire RAID Level 1. Note that the conceptual system has double the entire hardware configuration of a corresponding RAID Level 0 system, not just double the number of disks. Because of this, the cost of the resulting RAID Level 1 is exactly double the cost of the corresponding RAID Level 0 and dividing the resulting calculated RAID Level 1 throughput by two will always yield a fair throughput per cost figure.

This method, though superior in comparing throughput per cost, does not accurately model response time. We could assume response time is the same on both halves, as they do the same work. However, this ignores unsynchronized disks. Because of this difficulty in measuring response time, we choose to use
the previous method (constant number of data disks).

6. Experiment Implementation

Experiments were run on an Amdahl mainframe under UTS, which is a version of System V Unix. See Table 1 and Figure 4 for hardware statistics and channel architecture. Note that we have only one disk per string. This prevents channel conflicts and approximates a system which uses buffers to avoid data transfer conflicts.

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].

6.1. Process Structure

The simplest structure to produce a synthetic workload would be a single process issuing all I/O's. However, because UTS does not support asynchronous I/O, it is impossible to have more than one outstanding I/O per process. Thus, at the start of each experiment, one master process (the parent) creates one child process for each disk. This child process will be used as a user-level device driver and will drive one disk. All communication is done via IPC messages between the parent process and an individual child

<table>
<thead>
<tr>
<th>Processor Resources: Amdahl 5890-300e</th>
<th>Disk Resources: Amdahl 6380</th>
</tr>
</thead>
<tbody>
<tr>
<td>processors</td>
<td>cylinders/disk</td>
</tr>
<tr>
<td>cycle time</td>
<td>tracks/cylinder</td>
</tr>
<tr>
<td>memory size</td>
<td>sectors/track</td>
</tr>
<tr>
<td>data cache</td>
<td>bytes/sector (fixed format)</td>
</tr>
<tr>
<td>instruction cache</td>
<td>average seek</td>
</tr>
<tr>
<td>channels</td>
<td>average rotational latency</td>
</tr>
<tr>
<td></td>
<td>maximum transfer rate</td>
</tr>
<tr>
<td></td>
<td>disk caching</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>885</td>
</tr>
<tr>
<td>15 ns</td>
<td>15</td>
</tr>
<tr>
<td>256 MB</td>
<td>10</td>
</tr>
<tr>
<td>64 KB</td>
<td>4 KB</td>
</tr>
<tr>
<td>32 KB</td>
<td>15 ms</td>
</tr>
<tr>
<td>64</td>
<td>8.3 ms</td>
</tr>
<tr>
<td></td>
<td>2.4 MB/s</td>
</tr>
<tr>
<td></td>
<td>off</td>
</tr>
</tbody>
</table>

Table 1: Hardware Statistics of Processor and Disk Resources [Amdahl 6380, Amdahl 5890]. We assume rotational latency is distributed uniformly between 0 and 16.7 ms. We also assume seeks are distributed uniformly between 8 ms and 27 ms, an approximation of the seek time data in [Thisquen 88].
Figure 4: Disk - Memory Bus Architecture. This figure shows how the disks are connected to the memory bus in most of our experiments [Buzen 86]. Note that we use only one disk per channel path (path to memory). This avoids channel conflicts and RPS misses.

process—there is no communication between child processes. The child process has no concept of RAID Levels. Rather, the parent is responsible for generating the logical request, mapping the logical request into a number of physical requests, and passing the physical requests to the children. The children see simply a stream of disk accesses of the form: read/write, location on disk, size of access.

Each child process has a queue of requests, and waits until there is a request in its queue to carry out that request on its assigned disk (using UNIX read or write). When the request returns, the child will inform the parent of its return and process the next request in its queue. Note that no actual data passes between the parent and the child. All data is read into a garbage buffer and thrown away by the child. Data to be written also comes from a garbage buffer.

6.2. RAID Level 0

For a non-redundant array, individual logical reads and writes each result in a single disk access. Because there is no read buffering on the disks we are using, physical reads and physical writes have the
same disk service time. Consequently, in this experiment, we assume performance for RAID Level 0 is independent of the percentage of reads and writes.

6.3. RAID Level 1

To execute a mirrored read of a data block, the system must decide if the data disk or the shadow disk will carry out the access. In our experiment, we first look for a disk which is idle. If both disks are idle or both are busy, we choose the disk which will yield the shorter seek. To make this choice, we save the location of the previous request to that physical disk.

A possible optimization which do not make is to have a read serviced in part by a data disk and in part by a shadow disk. This implementation would increase throughput if the system was fairly idle, since it would make better use of both copies of each datum. However, it would decrease throughput for a
loaded system, since each disk would transferring less data per seek.

6.4. RAID Level 5

A RAID Level 5 write must update both the data blocks and the parity associated with those data blocks. Because a full stripe write will write an entire stripe, it involves all the data needed to compute parity, and no additional information needs to be read.

For partial stripe writes, two interesting implementation questions arise. First, which disks should we read to compute parity? One choice is to read the current data and current parity, compare the current data and new data, and change the parity correspondingly. When writing $D$ disks of data out of $N$ total disks, this choice results in $D+1$ disk reads to compute parity. The second choice is to read the data in the rest of the stripe and simply recompute parity of the stripe. This results in $N-D-1$ disk reads to compute parity. In our experiment, we choose the option that requires the fewest disk accesses. Partial stripes requests can range in size from 1 disk to $N-2$ disks. For an individual request, we read the current data and the current parity. This leads to a relative throughput of 25%. For a partial stripe write of $N-2$ disks, we read the remaining data disk and write the $N-2$ original data disks plus parity. This leads to a relative throughput of $\frac{N-2}{N}$ ($N$ disks need to be accessed in RAID Level 5 relative to $N-2$ disks in RAID Level 0).

Second, if we choose to read the current data and current parity, how do we schedule the new data and new parity writes? The new data can be written immediately after the current data has been read. This new data write will see a zero seek plus a full rotation. We assume the kernel can do the exclusive-ors necessary to compute the new parity block in the time the disk rotates once. Thus, we write the new parity block out one rotation after the current parity block has been read (exactly the same as the data block). This is a simplifying assumption, as the current data may not have been read yet. We believe this assumption is valid, as the device driver could simply delay the parity block read and write until the data block is about to be read. Because a request is not considered complete until all accesses related to that request have finished, such a minor scheduling delay would only marginally affect response time and should not affect throughput at all.
6.5. Response Time Control and Stabilization

To achieve a certain target 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 response time, and adjust the allowed number of outstanding logical requests accordingly. To avoid including the startup time in the performance analysis, we discard statistics gathered before the queue depth has stabilized. Queue depth stabilization in this context means being checkpointed at the same value more than twice. Once the queue depth has stabilized we begin collecting data. The throughput for the run is collected every 20 seconds. The run has stabilized when two conditions are met:

(1) The previous two throughput reports are within 1% of the current throughput report.

(2) 90%±1% of all requests have been fulfilled within the target response time.

Once the run has stabilized, we cease sending new requests and await the completion of any outstanding requests. While we await the completion of all requests, throughput will drop, though not enough to significantly affect the overall performance.

6.6. Synthetic Workload Implementation

The parameters of the synthetic workload are response time target, read/write ratio, request size distribution, and data distribution. When the parent process generates a request, it stochastically chooses 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. A second method is to choose a distribution of request sizes. We choose two distributions in particular: exponentials with small means and normals with large means and standard deviations. Once we choose the request size, we choose the placement of this data according to the data distribution.

Because there is no file system structure, we choose the data distribution by choosing the starting location of the logical request. In our workload, we break the starting location into two orthogonal
components: starting disk and stripe number on that disk. The starting stripe number on the disk is always distributed uniformly over all stripes on that disk.

We used several methods for choosing the starting disk. We call the first method *uniform alignment*. This is intended to be the data distribution which yields optimal performance. Uniform alignment 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.

The second method of choosing the starting disk is derived from a normal distribution (Figure 6). In our tests, the standard deviation (in disks) of our normal distribution equals the number of data disks. We refer to this data distribution as the *skewed* distribution.

![Graph showing the probability of being the starting disk](image)

*Figure 6: Using the Skewed Data Distribution to Select the Starting Disk.* The distribution of data accesses is determined by both starting disk and stripe number on that disk. The choice of starting stripe is always distributed uniformly over all stripes. Starting disk is chosen according to various distributions. The distribution shown here, the skewed data distribution, is the truncated right half of a normal distribution with a standard deviation of 10 disks.
7. Result Presentation

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 realistic as discussed in Section 4. Rather than jump from the analysis presented in the RAID paper directly to our most realistic workload, we make the transition in a number of smaller steps. These steps are summarized in Table 2.

We start by running the same workload as the RAID paper: request sizes are full stripe or individual; requests are all reads or all writes. In Section 8.1, we analyze an idle system to break down the time for a basic I/O. Next, in Section 8.2, we attempt to duplicate the assumptions made in the RAID paper of unlimited response time by analyzing a saturated system. In 8.3, we make the experiment more realistic 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 9.1, we remove the assumption of constant request size by using a distribution of request sizes. In 9.2, we skew the data distribution. As our last step in making the experiment more realistic, we allow workloads with both reads and writes.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Response Time</th>
<th>Request Size</th>
<th>Data Distribution</th>
<th>Read Write Mixture</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Breaking down a basic request</td>
<td>minimum</td>
<td>fixed</td>
<td>uniform aligned</td>
<td>unmixed</td>
</tr>
<tr>
<td>8.2</td>
<td>Unlimited response time</td>
<td>unlimited</td>
<td>fixed</td>
<td>uniform aligned</td>
<td>unmixed</td>
</tr>
<tr>
<td>8.3</td>
<td>Set target response time</td>
<td>target</td>
<td>fixed</td>
<td>uniform aligned</td>
<td>unmixed</td>
</tr>
<tr>
<td>9.1</td>
<td>Distribute request sizes</td>
<td>target</td>
<td>distributed</td>
<td>uniform aligned</td>
<td>unmixed</td>
</tr>
<tr>
<td>9.2</td>
<td>Skewed data distribution</td>
<td>target</td>
<td>distributed</td>
<td>skewed</td>
<td>unmixed</td>
</tr>
<tr>
<td>9.3</td>
<td>Mix reads and writes</td>
<td>target</td>
<td>distributed</td>
<td>skewed</td>
<td>mixed</td>
</tr>
</tbody>
</table>

Table 2: Stages of Results. A guide to the results in Sections 8 and 9. The workload becomes more realistic with each succeeding stage. In Sections 8.1-8.3, we concentrate on the response time target. In Sections 9.1-9.3, we change the request size, data distribution, and read/write ratio.
Lastly, we explore several additional issues, such as further varying the request sizes, comparing sector and track interleaving, and scaling the number of data disks. As a final caveat we look briefly at the effects of connecting multiple disks per channel.

Unless otherwise noted, experiments are run with 10 data disks and track striping. Thus, a full stripe is 10 tracks (400 KB) and an individual request is 1 track (40 KB).

8. The RAID Paper Workload

Request sizes in the RAID paper workload are large (a full stripe) or small (individual). 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, we use a very high workload and the uniform aligned data distribution.

8.1. Analyzing an Idle System

To understand the supporting hardware, 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. Average response time is not the 90% response time used in the majority of this paper, but rather the arithmetic average of all response times. We trace the lifetime of an average request by measuring the time spent in various stages of servicing the request (Figure 7). We also measure the total average response time of the requests and check that this is equal to the sum of the times spent in each stage. In all cases, the average response time is within 1 ms of the sum of the time spent in each stage.

Average response time breaks down as follows:

- **request overhead**: CPU time spent in sending messages between parent process and child processes.
- **IO CPU time**: CPU time for children 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**: time spent in seek and rotational latency. An average seek is 15 ms and the average rotational latency is 8.3 ms.
- **synchronization**: additional time due to multiple independent disks doing random seeks and rotations. This is not measured, but rather calculated (more in the next section) based on statistics given in Section 6.
- 19 -

- *connect time*: 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, pending time, and connect time are all measured *per disk access*.

8.1.1. Synchronizing Multiple Disks

There are two types of synchronization between multiple disks. 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, and is discussed in Section 8.1.4.

8.1.2. 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. Under normal loads, most of this overhead is overlapped with queuing time and does not noticeably affect performance. In an idle system, when the first of multiple messages is passed from the parent process to a child process, the message is immediately received and acted upon by the child (since the system is idle). As that child processes the I/O request it has just received, it contends for the CPU with the parent process, which is trying to send
Figure 7: 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.
out the other 9 requests. Thus, while sending out multiple messages, the parent process often has to wait through multiple context switches, as the child processes receive and act upon their messages. In contrast, in a non-idle system, most messages from the parent to the child processes are not acted upon immediately by the children (since they are usually in waiting for a previous I/O). As a result, the parent can finish sending the messages to the children in a short amount of time. 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 message passing overhead with more message passing channels. 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.

8.1.3. 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]

8.1.4. 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 7 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 7. For the first disk access (the read), we need to add synchronization because we are using two independent disks. However, RAID Level 5 individual writes are unique in that two disks that cooperate in one request do not necessarily cooperate in the next request. Thus, they 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 $\textit{seek + rotation}$. 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 $\textit{saving a seek}$.

8.2. Analyzing a Saturated System

As discussed in Section 6.5, we control the system load by controlling the number of outstanding logical request in the system. Increasing the target response time allows the system to have more outstanding logical requests at a time. This causes higher disk utilization and correspondingly higher throughput. By varying the load, we can graph absolute throughput per disk versus response time. Figures 8 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.

Note that the minimum response times are those discussed in Section 8.1. As expected, when we allow requests to have longer service times, we see higher throughput. Eventually, when the disks are fully utilized, increasing the response time no longer increases the throughput. The RAID paper analysis deals with this point of $\textit{maximum throughput}$. 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 9).

Figure 9 shows that, for most 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. 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
Figure 8: **Throughput vs. Response Time.** Throughput is graphed as a function of 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 response time. The data distribution used here is uniform aligned.
Figure 9: 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.

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 (total 72 ms). Applying the adjusted model to RAID Level 5 small writes, the total disk-time is approximately 72*2 = 144 disk-ms. The total disk-time for RAID Level 0 small writes is approximately 40. 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.
8.3. Maintaining Equal Response Times

Section 8.2 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 with different target response times. By forcing all RAID Levels to have equivalent response times, we generate a fairer comparison. Figure 8 showed that throughput for all RAID Levels drops as the response time target decreases. To better understand how throughput changes for each RAID Level, we graph throughput per disk versus 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 Figures 10a. 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 10a.

These response time behavior figures show us what to expect when we maintain equal response times for each RAID Level. For example, in Figure 10a, we see that at any response time less than 1000 ms, RAID Level 5 achieves a lower percentage of its maximum throughput than RAID Level 0. This is due to RAID Level 5 small writes having long response times as discussed in Section 8.1.4. Thus, RAID Level 5's relative throughput (relative to RAID Level 0) for small writes will decrease. In contrast, RAID Level 1 small writes track RAID Level 0 small writes very closely at all response times. Thus we expect RAID Level 1 small writes to maintain the same relative throughput.

In the remainder of this paper, we choose one specific response time for each workload. Our method for choosing that response time is somewhat arbitrary. We first measure the minimum response time of a RAID Level 0 (as in the idle system in Section 8.1) running a particular workload. The target response time for that workload is then set at four times that minimum response time. To summarize, 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 response time is 268 ms; for individual requests, target response time is 200 ms. This is intended to allow some freedom to queue requests in order to achieve higher throughput, but not to allow queues to grow too deep. Notice that four times the idle RAID
Figure 10: 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 response time target. For example, in Figure 10a, RAID Level 5 achieves a lower percentage of its maximum throughput than either RAID Level 0 or 1. Data distribution is uniform aligned.
Figure 11: Before and After Equalizing Response Times. Throughput per disk relative to RAID Level 0 is shown before and after maintaining equal response times. We see that equalizing to our target response times affects relative throughput for RAID Level 1 individual reads and RAID Level 5 individual writes.

Level 0 response time for large requests easily exceeds the response time needed to achieve maximum throughput. An interesting future experiment would be to restrict large requests to a small percent over the minimum response time.

Figure 11 shows the relative throughput per disk of RAID Levels 1 and 5 before and after equalizing to these target response times. Note that at the particular 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 response times. Similarly, Figure 10b shows that, at our 200 ms target response time, RAID Level 1 has is at a slightly higher percentage of its maximum throughput.
This section has focused on response times. We have seen that limiting response times causes less than maximum throughput for all RAID Levels. In general, RAID Levels with longer response times are penalized more than RAID Levels with shorter response times.

9. More Realistic Workloads

In Section 8, we analyzed the RAID paper workload of full stripe or individual requests, 100% reads or 100% writes. We started by analyzing idle systems (minimum response time) and saturated systems (maximum throughput). We finished by analyzing systems with equivalent response times. While still maintaining equivalent response times, we now begin modifying the actual workload: request size, data distribution, and read/write ratio.

9.1. Distributing Request Sizes

A key characteristic of any workload is the logical requests sizes. Until now, large requests have been full stripe requests and 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. Rather, they issue requests of various sizes. 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 response times will follow the changes in RAID Level 0 minimum response times.
Figure 12: Effect on Absolute Throughput Of Changing Request Sizes. Throughput after distributing request sizes is shown as a percentage of the throughput before distributing request sizes. Throughput for large requests improved to about 160%; throughput for small requests decreased to about 25% of the undistributed request sizes.

(new response time target for large requests will be 780 ms; new response time target for small requests will be 148 ms).

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 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 for all RAID Levels. These trends in absolute throughput are shown in Figure 12, which shows throughput after distributing request sizes as a percentage of the throughput before distributing request sizes.
The same general trends hold for all RAID Levels: throughput increases for large requests and decreases for small requests. However, there is some variation in how much each RAID Level changes. To better picture how the performance of RAID Levels 1 and 5 change relative to RAID Level 0, we plot the new relative throughput per disk. 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, shown in Figure 13.

![Graph showing relative throughput for RAID Levels 1 and 5.]

Before distributing request sizes
- R,W: request size = full stripe = 400 KB
- r,w: request size = individual = 40 KB

After distributing request sizes
- R,W: request size = normal distribution, mean 1.5 MB
- r,w: request size = exponential, mean 6 KB

**Figure 13: 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 response time target. Data distribution is uniform aligned.
9.1.1. 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 turns out to be the same.

9.1.2. RAID Level 5 Small Writes

As shown in Figure 13, 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 13.

Most of the improved relative throughput is due to selecting a response time target which is more favorable for RAID Level 5 small writes than the response time target for undistributed request sizes. Figure 14 is similar to Figure 10a characterizing the throughput/response-time profile for small writes. In Figure 10a, throughput at the target response time (200 ms) was at 59% of maximum throughput for RAID Level 5 and 81% for RAID Level 0. In Figure 14, throughput at the target response time (148 ms) remains roughly the same for RAID Level 0 (85%); However, 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 is higher than in Section 8.3.
Figure 14: Percent of Maximum Throughput for Small Writes. Similar to Figure 10, throughput per disk for each RAID Level is shown as a percentage of the maximum throughput per disk for that RAID Level. Size distribution is exponential with a mean of 6 KB. Note that the target response time of 148 ms is more favorable for RAID Level 5 than the target response time in Figure 10a. Data distribution is uniform aligned.

9.1.3. RAID Level 5 Large Reads

For full stripe reads, the relative throughput of RAID Level 5 was 100%. This result was expected, as no extra redundant information needed to be read in, and the number of disk accesses in the physical request equaled the number of disk accesses in the logical request. However, now that requests cover multiple stripes, we can no longer simply read the data and ignore parity. For example, in Figure 1b, 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 child process 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 13.

With tighter control of the channel program, we could build one channel program to issue multiple I/O's. This would save on turnaround time between the two I/O's and prevent the missed revolution. Another solution is to map the data such that the first data sector on a track after a parity track starts several sectors past the previous data sector.

9.1.4. 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 1b, 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 6.4, partial stripe writes can have relative performance ranging from 25% to \( \frac{N-2}{N} \).

In this section, we allow at most one partial stripe per request. This is done by using the uniform aligned data distribution (see Section 6.6). 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 other than uniform aligned (as in the following section), we see up to two partial stripes per request, and a correspondingly higher drop in relative throughput.

9.2. Varying the Data Distribution

Because we interleave data across disks in fixed units, hot spots tend to be spread among several disks and naturally smoothed out. Realistically, however, some disks will still receive more requests than
Figure 15: Change in Absolute Throughput After Skewing Data Distribution. Throughput after skewing the data distribution is shown as a percentage of the throughput before skewing the data distribution. Note that skewing the data distribution has little effect on throughput for large requests, whereas throughput for small requests decreases by approximately 10%.

In this section, we remove the assumption of uniform disk utilization by using a skewed data distribution. We continue to maintain equal response times. We also continue to use the most recent definitions of large and small requests (large is defined to be a normal distribution of request sizes with a mean of 1.5 MB; small is defined to be an exponential distribution of request sizes with a mean of 6 KB).

As discussed in Section 6.6, skewing the data distribution reduces to choosing the starting location (starting logical disk and starting stripe) of the logical request. The starting logical disk and the starting stripe on that disk are chosen independently. In a skewed distribution, the starting disk is chosen according to a normal, with standard deviation equal to the number of data disks (Figure 6). The starting stripe is always chosen according to a uniform distribution across all stripes.
Figure 15 shows the throughput after skewing the data distribution as a percentage of the throughput before skewing the data distribution. For all workloads, skewing the data distribution causes the absolute throughput to decrease. In most cases, the throughput for large requests only decreases a few percent. Because large requests cover multiple stripes, choosing the starting disk of the request generally has little effect on overall throughput. An exception to this is RAID Level 5 large writes (discussed below), whose throughput decreases more than RAID Levels 0 or 1. Small requests show a larger (10%) decrease in throughput. An exception is again RAID Level 5, whose throughput decreases only slightly.

![Graph showing throughput data](image)

**Figure 16: Relative Throughput Before and After Skewing the Data Distribution.** This graph shows the effect of skewing the data distribution on relative throughput. The data distribution was changed from uniform aligned to skewed. RAID Level 5 small reads improved slightly due to the location of the parity tracks. RAID Level 5 large writes degraded due to additional partial stripe writes.
RAID Level 5 large write throughput decreases significantly because of additional partial stripe writes. When distributing the request sizes, a number of partial stripe writes were introduced. Using the uniform aligned data distribution, each request had at most one partial stripe. Now, with the skewed data distribution, no attempt is made to align the request on a full stripe boundary. Thus, up to two partial stripe writes can occur per request and throughput decreases accordingly.

For small requests, the general trend when skewing the data distribution is a decrease in throughput of 10%. However, RAID Level 5 throughput stays relatively constant. There are several reasons for this difference.

First, the skew of the data distribution is done on the logical disk. In RAID Level 0 the logical disk is identical to the physical disk. In RAID Level 1 the logical disk maps to one or both of two physical disks, independent of which stripe is accessed. However, for RAID Level 5, a logical disk does not map into a single physical disk. Rather, the mapping from logical disk to physical disk depends on which stripe is accessed. For example, in Figure 1b, logical disk 0 (stripes 0, 4, 8, etc.) is spread over physical disks 0 and 1. Thus, the skew on the data distribution is smoothed over by the logical to physical mapping done by RAID Level 5. This lessens the effect of skewing the data distribution for both RAID Level 5 small reads and small writes.

For RAID Level 5 small writes, the skewed data distribution is further smoothed because two disks are involved in each request. Recall that the starting stripe is always chosen uniformly over all stripes. Thus, although the choice of the data disk is non-uniform, the choice of the parity disk, which depends on both the starting disk and the starting stripe, is uniform over all disks. Having the parity disk uniformly chosen from all disks again smooths the skew on the disks.

We again plot the relative throughput for each RAID Level, both with and without the data distribution skewing (Figure 16).

9.3. 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 response times, distributed request sizes, and skewed data
distributions, we at last mix reads and writes.

At first thought, we may expect the throughput for RAID Levels 1 and 5 to be the weighted average of the throughputs for 100% reads and 100% writes. This would cause linear variation in Figure 17. Instead, though close to linear, throughput changes superlinearly with the fraction of reads in the workload. To understand why throughput is superlinear, consider an idle system receiving a stream of read or write requests. Assume reads have a response time of $R$ and writes have a response time of $W$. Estimating throughput as the reciprocal of the average response time, the throughput for 100% reads is $\frac{1}{R}$ and the throughput for 100% writes is $\frac{1}{W}$. A mix of 50% reads and 50% writes will not have a throughput that is

![Graphs showing throughput for different RAID levels]

Figure 17: 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 (response time target of 780 ms). The size distribution for small requests is an exponential with a mean of 6 KB (response time target of 148 ms). Data distribution is skewed.
the weighted average between reads and writes, i.e., \( \left[ \frac{1}{R} + \frac{1}{W} \right] \). Rather, because the average response time is \( \frac{R + W}{2} \), the throughput of the system should be \( \frac{2}{R + W} \). Estimating throughput as the average of the constituent throughputs (averaging the reciprocal of the response times) will always be higher than the actual throughput (the reciprocal of the average response time). Thus, a graph of throughput against fraction of reads will be superlinear.

9.4. Summary of More Realistic Workloads

To summarize, we look at Figures 17. Note that for small requests, RAID Level 1 consistently yields higher throughput per disk than RAID 5. In fact, with over 90% reads, seek optimization allows RAID Level 1 to yield higher throughput per disk than even RAID Level 0.

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.

10. Additional issues

In this section, we explore additional issues, such as further varying the request size distribution, varying the unit of interleaving, and scaling the number of disks. Unless otherwise stated, we define large as a normal distribution with mean 1.5 MB and small as an exponential distribution with mean 6 KB. We also continue to equalize response times and skew the data distribution. A major change from the previous section is that we now define the read workload to be a mixture of 90% reads and 10% writes; similarly, we now define the write workload to be a mixture of 90% writes and 10% reads.

10.1. Varying the Request Sizes

In Section 9.1, we both distributed the request sizes and changed the mean. In this section, we continue to change the mean of the request size distribution by using various normal distributions.

Figure 18 graphs the relative throughput for RAID Levels 1 and 5 against average request size. In Figure 18a, RAID Level 1 relative throughput for writes is approximately 50% for all sizes, as expected. In contrast, relative throughput for RAID Level 5 writes increases as request size increases. As requests
become larger and cover more full stripes, RAID Level 5 becomes dominated by full stripe write performance rather than the partial stripe write performance and relative throughput approaches $\frac{N-1}{N}$. Below .5 MB, RAID Level 1 has higher throughput per disk than RAID Level 5; at sizes larger than .5 MB, RAID Level 5 yields higher throughput per disk than RAID Level 1.

In Figure 18b, RAID Level 1 relative throughput for reads decreases with increasing request size. With small requests, seek optimization pushes RAID Level 1 relative throughput above 100%. As requests get larger and response time increases, the benefit of seek optimization is minimized. Relative throughput then drops to 90%. The 10% writes in the workload cause the total relative throughput to be less than the read performance in Section 9.2 (120% for small requests, 100% for large requests).

For RAID Level 5, relative throughput is 85% for small request sizes, then increases to 90%-95% as request sizes increase. This effect is due almost entirely to the 10% writes in the workload. For small
request sizes, writes have a relative throughput of 20%-25%. However, at larger request sizes, the relative throughput of writes is 70%-80%, so relative throughput increases.

10.2. Sector Stripping vs. Track Stripping

Up to now, we have always used track striping. In this section, we explore sector level striping. We address two questions:

- Is it possible to analyze RAID Level 1 and 5 independent of the unit of interleaving?

![Figure 19: Relative Throughput for Track and Sector Stripping](image)

The relative throughput per disk with sector striping is compared against the relative throughput per disk with track striping. We see that using sector striping only affects relative throughput for RAID Level 5 writes. The response time target for small requests is 148 ms; the response time target for large requests is 780 ms. The read workload is 90% reads and 10% writes; the write workload is 90% writes and 10% reads. Data distribution is skewed.
- Which unit of interleaving gives the best absolute throughput?

The RAID paper analysis was independent of the unit of interleaving. Dependence from unit of interleaving, as well as from factors such as hardware specifications, was eliminated partly by evaluating the throughput relative to RAID Level 0. Also, the analysis achieved independence from the unit of interleaving by basing the analysis on full stripe or individual requests. These request sizes scaled with the unit of interleaving. In our experiment, we fix the request size distribution independent of the unit of interleaving to learn if identical workloads can be analyzed without regard for the unit of interleaving. Figure 19 shows the relative throughput for RAID Levels 1 and 5 for various sizes with 90% writes and 90% reads.

![Diagram](image)

**Figure 20: Change in Absolute Throughput after Changing from Track to Sector Striping.** Throughput with sector striping is shown as a percentage of the throughput with track striping. Throughput with track striping is generally better than throughput with sector striping. An exception to this is RAID Level 5 large writes. The reads workload is 90% reads and 10% writes; the write workload is 90% writes and 10% reads. Data distribution is skewed.
For RAID Level 1, relative throughput with sector striping is close to relative throughput with track striping. Similarly, for RAID Level 5 reads, relative throughput with sector striping closely matches relative throughput with sector striping. However, for RAID Level 5 writes, the relative throughput changes with the unit of interleaving. Thus, although analysis for RAID Level 1 and RAID Level 5 reads can be done with little regard to unit of interleaving, the analysis for RAID Level 5 writes should take into account the unit of interleaving.

To compare sector striping to track striping in terms of absolute throughput, we show throughput with sector striping as a percentage of throughput with track striping (Figure 20). The general trend in Figure 20 is that changing from track to sector striping causes throughput to decrease. This decrease is more pronounced for small requests than for large requests. For very small requests (single sector), the disks see the same stream of physical requests using either sector striping or track striping. Slightly larger requests (larger than a sector but less than a track) are mapped onto multiple disks with sector striping but onto a single disk with track striping. Thus, in track striping, the single disk is transferring more data per seek than the multiple disks are in sector striping. This causes lower throughput with sector striping. With extremely large requests (more than 10 tracks), both sector striping and track striping cause disks to transfer the same amount of information, leading to roughly the same throughput.

The sole exception to this trend in changing from track to sector striping is RAID Level 5 writes. The trend of decreased throughput with sector striping was caused by spreading a request over more disks, with each disk transferring less data. However, RAID Level 5 writes benefit by using more disks in servicing a request. Relative throughput for partial stripe writes ranges from 25% for single disk partial stripe writes to \( \frac{N-2}{N} \) for wider (many disks) partial stripe writes. Thus, spreading requests over more disks leads to wider partial stripes and higher relative throughput. This can offset the performance degradation caused by having each disk transfer less data. With very large requests, throughput also increases because the partial stripe portion of the request will be a lesser fraction of the total request.

10.3. Scaling the Number of Disks

So far in this paper, we have limited the maximum number of total disks to 20. All RAID Levels have used 10 data disks, with RAID Level 1 using 20 total disks and RAID Level 5 using 11 total disks. In
this section, we explore what happens as we vary the number of data disks used.

Figures scale shows how the throughput per disk varies as we vary the number of data disks. We continue to use a target response time of four times the response time of an idle RAID Level 0 system. We also continue to define small as an exponential distribution with mean 6 KB. However, we change our definition of a large request. With 10 data disks, our previous definition of a large request (normal with mean 1.5 MB) covered an average of 4 stripes. Thus, each disk transferred 4 tracks per seek. However, with more data disks, this average sized request no longer covers 4 stripes. To compensate, we scale the size of an average request along with the number of disks. We maintain an average request size of 150 KB/data disk. For example, with 20 data disks, we use an average request size of 3 MB. With this modification, an average large request will continue to cause each disk to transfer approximately 4 tracks.

In general, throughput per disk is approximately constant, showing that it is independent of the number of data disks. An important exception occurs when the total number of disks exceeds 20. For example, a RAID Level 1 system with 15 data disk (30 total disks) shows a dramatic drop in throughput per disk. As discussed in Section 5.2, this drop in throughput per disk comes from not scaling the CPU power along with the number of disks. With more than 20 disks, the CPU begins to limit performance and maintaining constant throughput per disk becomes impossible without more CPU power.

We also see that the throughput per disk of RAID Level 5 large writes (Figure 21c) increases slightly with more data disks. This is because there is always one parity disk per system, and, with more data disks in the system, the overhead of updating this parity disk affects overall system performance less. Thus, throughput per disk increases.

10.4. Multiple Disks per Channel Path

We have, so far, connected one disk per channel path (Figure 4). When a disk is ready to transfer information, no channel conflicts are possible. With more than one disk is connected to a channel path, (Figure 22) channel conflicts are possible [Ng 88]. These channel conflicts delay the disk from transferring data and cause the disk to miss a rotation (an RPS miss). Figure 23 shows the effect of connecting multiple (one or two) disks per channel path, by showing throughput with multiple disks per channel path as a fraction of throughput with one disk per channel path.
Figure 21: How RAID's Scale. Throughput per disk is shown as a function of the number of data disks. In general, throughput per disk is constant under 20 total disks. The size distribution for large requests is a normal with a mean of 150 KB per data disk. The size distribution for small requests is an exponential with a mean of 4 KB. Response time target for small requests (for all numbers of disks) is 148 ms. Response time target for large requests: 5 data disks: 720 ms, 10 data disks: 780 ms, 15 data disks: 876 ms, 20 data disks: 924 ms. Data distribution is skewed.
Figure 22: Architecture with Multiple Disks per Channel Path. This figure shows how multiple disks per channel are connected to the memory bus for Section 10.4. Strings of four disks are shared between two paths to the memory bus.
Figure 23: Change in Absolute Throughput after Connecting Multiple Disks per Channel Path. Throughput with multiple disks per channel path is shown as a percentage of throughput with one disk per channel path. Throughput for large request sizes drops to half the throughput with one disk per channel path, whereas throughput for small requests only decreases 10%.

Two channel paths are connected to one string of four disks. We represent a string by letters and the disks on a string by numbers (e.g. disks a0, a1, a2, a3 make up one string). RAID Levels 0 and 5 have a straightforward mapping: disks 0-10 are a0-a3, b0-b3, c0-c2. Thus, RAID Levels 0 and 5 have 10 or 11 disks with 6 channel paths total. RAID Level 1 has the following mapping: the primary data disks are a0-a3, b0-b3, c0-c1; the shadow disks are c2-c3, d0-d3, e0-e3.

Throughput for large requests drops to 50%-55% of the throughput with one disk per channel. Because each disk is transferring an average of 4 tracks per request, the channel is often busy. Sharing a busy channel causes a severe drop in throughput. Small requests, on the other hand, only drop in throughput by 10%-15%. Because very little time in a small request is spent transferring data (and thus tying up a channel path), little potential for channel conflicts exist and little penalty is seen.
11. Summary

We have started with a simple model for performance and, step by step, measured performance with more and more realistic experiments. 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 response times. We found that equalizing response times hurt the relative throughput of RAID Levels with higher response times (RAID Level 5 small writes) and helped the relative throughput of RAID Levels with lower 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.

Fourth, we mixed reads and writes. We found that RAID Level 5 had higher throughput per disk than RAID Level 1 for large requests for almost all mixes of reads and writes. In contrast, RAID Level 1 had higher throughput per disk than RAID Level 5 for small requests for all mixes reads and writes.

Lastly, we explored issues such as varying the request sizes and using sector striping. We found that track striping was usually better than sector striping, with the notable exception of RAID Level 5 large writes.

12. Future Work

We are continuing to analyze the performance of disk arrays. In particular, we are interested in arrays of small disks. We are designing, building, and evaluating a disk array of 30-50 CDC Wren 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.
13. Acknowledgements

First and foremost, "thanks be to God, who always leads us in His triumph in Christ. (2 Cor 2:14)". Also thanks to Donna, Andy, and the rest of Crusade for their constant encouragement.

Technically, many of the research ideas are due to Garth Gibson, who spent many hours discussing, advising, and mothering throughout. Dave Patterson, Susan Eggers, and Alan Smith also provided much needed advice. Thanks also go to Randy Katz and Dick Wilmot for their helpful reviews. At Amdahl, John Colgrove, Art Casabal, and Romeo Narciso were indispensable resources.

The Amdahl Corporation generously provided many hours of expensive standalone time. I was supported in part by an ONR fellowship. The RAID project is funded in part by the National Science Foundation under grant #MIP-8715235. The RAID project would also like to thank Sun Microsystems Incorporated, Impress/CDC, and other industrial partners for their support.

14. Bibliography


The following is the data used in this paper:

Field 1 (eq. r0, r1, r2): program used; r0 = raid0, r1 = raid1, r2 = raid6
Field 2 (eq. t0): unit of interleaving, in 4K sectors
Field 3 (eq. t1): code for read/write ratio:
    t7 = 70% reads
    t8 = 10% reads
    t9 = 100% writes
    t10 = 70% reads
    t11 = 50% reads
    t12 = 70% reads
    t12 = 70% reads
    t13 = 100% reads
Field 4 (eq. s1): code for request size distribution
    s1 = full stripe
    s2 = individual
    s3 = normal with mean 100 sectors, standard deviation 100 sectors
    s4 = exponential with mean 1 sector
    s5 = normal with mean 450 sectors, standard deviation 450 sectors
    s6 = normal with mean 150 sectors, standard deviation 150 sectors
    s7 = normal with mean 25 sectors, standard deviation 25 sectors
    s8 = normal with mean 100 sectors, standard deviation 100 sectors
    s9 = normal with mean 500 sectors, standard deviation 500 sectors
    s10 = normal with mean 600 sectors, standard deviation 600 sectors
    s11 = normal with mean 192 sectors, standard deviation 192 sectors
Field 5 (eq. t10): code for data distribution
    15 = normal with standard deviation 15 disks
    16 = normal with standard deviation 10 disks
    17 = normal with standard deviation 5 disks
    18 = normal with standard deviation 3 disks
Field 6 (eq. lat-5, lat-200): target response time (for 90% latency marks)
    lat-5 = keep the queue depth timed at x outstanding logical request
    lat-200 = target the response time at xxx ms
Field 7 (eq. d10): number of disks
Field 8 (eq. r0.383): request overhead, in ms
Field 9 (eq. l123): measured response time (90% marks)
Field 10 (eq. l-999): measured percentage of requests which met the measured response time
Field 11 (eq. l-81): measured percentage of requests which met the target response time
Field 12 (eq. l-184): measured average response time in ms
Field 13 (eq. l-1399): number of I/O's
Field 14 (eq. e0): amount of CPU time spent in exclusive or
Field 15 (eq. s100): measured average size of request
Field 16 (eq. s300): measured standard deviation in size of requests
Field 17 (eq. t160255.6): total time of run
Field 18 (eq. th9.062298): throughput in MB/s
Field 19 (eq. t120.201728): I/O's per second
Field 20 (eq. er40.0411): total time per access spent by child process
Field 21 (eq. chol.27629): cpu time spent per access by child process
Field 22 (eq. q5.1788): average queue depth over the run
Field 23 (eq. ssch3993): total number of start subchannels
Field 24 (eq. conn16.1293): average connect time per disk access
Field 25 (eq. disc22.5913): average disconnect time per disk access
Field 26 (eq. pend.23763): average function pending time per disk access
Field 27 (eq. cpu2912.72): total cpu time
Field 28 (eq. u0): run reached stability (u1) or exit (u0)