Storage System Metrics for Evaluating Disk Array Organizations

Randy H. Katz[1]
Electrical Engineering and Computer Science Department
University of California, Berkeley
Berkeley, CA 94720
and
SF2 Corporation
Sunnyvale, CA

David W. Gordon
Array Technologies Corporation
Boulder, CO

James A. Tuttle
J. A. Tuttle Associates
Los Gatos, CA 95030

Abstract: Disk arrays represent a new way to construct high performance I/O systems, by replacing a few large formfactor drives by many small drives. Data is striped across the drives, with data redundancy introduced to improve media availability. A fundamental issue is how best to configure the array for performance, capacity, and availability. We examine three possible array organizations: no redundancy ("simplex" or RAID Level 0), 100% redundancy ("mirroring" or RAID Level 1), and parity redundancy (RAID Level 5). The mirrored approach favors performance over capacity, while the opposite is true for parity protected arrays. Since no single array organization clearly dominates the others, we present several alternative metrics by which to compare them. We argue that a way to simultaneously evaluate capacity and performance is needed, and propose a new storage system metric based on data temperature (IOs/second/GB) for this purpose. We illustrate its use by assessing several example array organizations.

1. Introduction

Recent advances in magnetic storage device technology are having a dramatic effect on storage system organizations. Perhaps the most significant trend has been the steady decrease in magnetic disk form factor, from 14" large drives to 8"- 10" midsized drives to 5.25" - 3.5" small drives, coupled with a remarkable increase in storage capacity. In 1990, it is possible to purchase 5.25" disk drives that store 1.7 GBytes of data [Voelcker87] and transfer at a rate of 3 MBytes/second, and 3.5" drives with a 320 MBytes capacity and a 1.25 MBytes/second transfer rate [IBM89]. This is essentially comparable to older generations of 8" [Fujitsu87] and 14" [IBM87] drives.

Because of the technological factors that drive the disk industry, storage capacity has significantly improved and will continue to do so, doubling every two to three years. Transfer rates are also increasing, albeit more modestly, as drives spin faster (the rotation rate has increased from 3600 rpm to 5400 rpm), as track recording densities rise, and as zoned bit recording techniques with their ability to store more bits on the outer tracks becoming more pervasive.

In addition to disk transfer rate, disk service time consists of queuing delay, seek time, and rotational latency. The faster spin rates help to improve the service time, by reducing rotational latencies (from 16.7 ms at 3600 rpm to 11.1 ms at 5400 rpm), but seek times, from 12 to 16 ms for the "average" seek, improve only moderately. Despite the advances in spin rates and recording density, a typical disk can perform 30 to 50 random I/Os per second per read/write actuator,

1. First author's research supported by the National Science Foundation, the Defense Advanced Research Projects Agency (DARPA), the National Air and Space Administration (NASA), and the California MICRO Program in conjunction with DEC, Emulex, Kodak, IBM, Intel Scientific, NCR, and StorageTek.

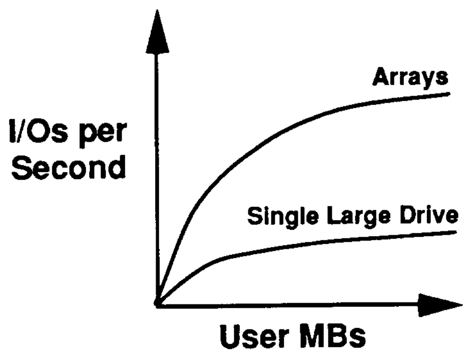

Figure 1: Arrays Deliver More Actuators per MB
By using small formfactor drives, a disk array can provide more actuators per MB, thus spreading I/O requests over more actuators and thereby reducing queuing delays and user latency. In addition, the approach provides for more parallel pathways to data on disk, and thus can result in increased bandwidth to the disk system.

independent of the formfactor. Given that processors are doubling in their MIPS rating every year or two and that the computing environment is moving towards multiprocessors and/or "clusters" of processors sharing a common file or storage service, it is easy to see that I/O rates present a formidable bottleneck. Many believe that the critical "I/O gap" will seriously limit future system performance.

Small format disk devices make attractive a new storage system organization, the *disk array*. For a given amount of storage capacity, a disk array obtains high performance by replacing a few large format disk drives with many small format drives (see Figure 1). The effect is to replace a few disk actuators with many disk actuators, making possible a much higher aggregate data and I/O rate. For this to be realized by a given user application, its files must be spread across these drives, a technique often called "striping" [Salem86]. A significant disadvantage of striping is that if a single disk within the stripe set fails, none of the data can be accessed. A single failure renders the array *unavailable*. Since the failure rate of a system scales up with the number of components, a storage system with many disk drives will experience a disk failure (or controller failure or power supply failure, etc.) fairly often. Thus, it is critical that a disk array include some mechanism for insuring that even if a disk drive fails, the data within the array is still accessible. A common technique is to store the data more than once, usually in an encoded form, so that the data on a failed disk can be reconstituted from redundantly stored information within the array. Therefore, the attributes of a storage subsystem that distinguish a disk array from a "disk farm" are:

- many small physical disk drives to achieve a high aggregate I/O and data rate
- files striped over multiple disk actuators
- data redundancy for high availability

The rest of this paper is organized as follows. In the next section, we describe two kinds of redundant array organizations: mirrored disks (sometimes called a *2N array*) and parity protected arrays (called *N+1 arrays*). In Section 3, we discuss storage system "figures of merit" in widespread use today, and point out why they are inadequate for evaluating alternative array configurations. Section 4 presents a metric that combines aspects of performance and capacity, called *data temperature*. The ultimate goal is to be able to compare system reliability within the same framework, but this must be saved for a future publication. The application of this metric to several alternative storage configurations is given in Section 5. Our summary and conclusions are in Section 6.

2. Mirrored Disks and Parity Array Organizations

It has long been known that the primary approach for obtaining high availability storage subsystems is through *redundancy*. Data is stored in such a fashion that each bit of data contributes to another redundant bit stored elsewhere within the array (some detailed mechanisms are described below). The concept of redundancy is not limited to data; it can — indeed, in a true fault tolerant system, it must — also apply to the paths to data (controllers and cabling) as well as support hardware such as power supplies and fans. Disk system organizations that store redundant data for the purpose of achieving high availability are called RAIDs, or *Redundant Arrays of Independent Disks.*

There are two intrinsic penalties associated with any method that uses data redundancy to improve availability: first, storage capacity is sacrificed to store the extra redundant data, and second, write bandwidth is forfeit to keep it up to date. There are several alternative RAID organizations that represent different tradeoffs between availability, I/O performance, and the capacity penalty [Patterson88]. The two array organizations best suited for high availability and high I/O rate are *mirrored disks* (also known as 2 N arrays) and *parity arrays* (sometimes called N+1 arrays). Each organization is described next.

In a mirrored disk organization, each disk containing user data is associated with a mirrored partner. Each data bit on one disk is stored redundantly on the other. Thus, the capacity overhead for redundancy is 100%. It is no surprise that I/O performance is also affected by the redundancy. Since each disk "mirrors" the other, data can be read from either but writes must be reflected to both. Thus, the write rate is one–half of what a single disk can deliver (i.e., the write rate of the pair is equal to the number of writes that can be sustained by a single disk drive). Because the data is placed under two independent actuators, the read rate is slightly better than what one disk can provide because of a technique called *seek scheduling*. That is, the disk with its head closest to the requested data is allocated to service that request. Improved read rates of 15–20% have been observed, although the results depend critically on the incidence of reads versus writes in the stream of requests [Chen90]. If one member of the pair fails, then the surviving member services all requests until the other is replaced. Interestingly enough, write performance is likely to improve while read performance is made worse in this degraded state. However, assuming a reasonably short repair time, such as a 72 hour/3 day repair contract, the probability that both members of the pair will be broken at the same time is quite small, and thus a very large expected mean time to datra loss (MTTDL) can be achieved.

The parity array organization employs a more clever method to encode redundant data, dramatically reducing the capacity overhead. The array is partitioned into independent *recovery groups* within which a simple redundant encoding is performed. In mirroring, the recovery group size is two, that is, each calculation of redundancy applies to exactly two disks, and any two disk group can survive a single failure. (A disk system constructed from multiple pairs can *suffer a single failure in each pair* before it is vulnerable to data loss). In parity arrays, the size of the group is *N+1*, that is, *N* data disks plus an additional disk's worth of capacity to hold redundant data.

Redundant data is introduced into the array by computing parity bit by bit horizontally across the disks that form the recovery group. K bits are of *odd parity* if the number of "1" bits is odd. Even parity is described analogously. N bits, one from each of the remaining data disks within the group, are combined to provide a single N+1st encoded parity bit. For odd parity, if the N bits contain an even number of 1's, then the N+1st bit is set to 1, making the entire string of bits odd. If the N bits contain an odd number of 1's, then the N+1st encoded bit is set to 0. If any single disk fails within the recovery group, a given bit on the failed drive can be reconstituted from the associated bits on the surviving N disks, maintaining the appropriate sense of the parity: an odd number of 1's means the failed disk must have stored a 0, while an even number implies it contained a 1. If hot spares are included within the array, accesses can continue to the failed disk while its contents are being reconstructed onto a spare. With this scheme, the "window of vulnerability" to a second failure within a group can be shrunk from the usual 72 hour repair call to 1 – 2 hours, significantly increasing the MTTDL. Also with this approach, the storage overhead to obtain tolerance to single failures within a group is reduced to 1/(N+1) bit of redundancy for each data bit. This can be made arbitrarily small simply by increasing group size.

Unfortunately, sector writes become more complicated in this organization. For each bit that is written, the associated parity bit must also be updated. To do this, it is necessary to read the old data bits, compute the difference between the old and new bits, read the old parity bits, and complement any of them that are associated with changed data bits. The computation is actually quite straightforward, but it is easy to see that a logical write becomes four physical I/Os: read old data, read old parity, write new data, write new parity. Thus, the write rate is reduced to 25% of what a non-redundant disk system could deliver. (This "read before write" overhead can be avoided if data is written in units that are large enough to span the entire recovery group — however, this is only possible for applications such as scientific codes that read and write large data sets in bulk.) It is a general observation for parity arrays that the capacity overhead is significantly reduced at the expense of an increased penalty for "small" writes.

The reader may infer from the description above that all parity bits are placed on a single N+1st disk. This need not be the case, and in fact, represents a serious bottleneck since all writes to the group would then update the same parity disk (a single disk can perform no better than 30 – 50 random I/Os per second). It is a straightforward generalization to compute parity in larger units, such as a track or cylinder. These can be interleaved among the disks of the group so that the parity blocks are evenly distributed among the members of the recovery group.

To summarize, mirrors have a higher I/O rate per actuator, but much worse user capacity than a parity array. Disk arrays offer better capacity, but a lower I/O rate. Mirrors offer better availability than arrays, but the difference disappears when hot spares are added to arrays (see [Gibson91] for details). Given the possible tradeoffs between I/O performance, user capacity, and availability, and the different system workloads that are possible, the major question is to determine the most economical way to configure a storage system with very many disks.

3. Storage System Metrics

There are many different ways of describing the cost, capacity, performance, and availability of a storage system. But unfortunately, there is no single metric that blends together these disparate aspects.

The appropriate metric is highly dependent on the application and the user environment. In a time sharing environment, capacity cost ($/MByte) may be the overriding consideration. The performance of an image processing application will be most sensitive to the disk transfer rate (MBytes/second), while the I/O processing rate (IOS/second) will be most relevant for a commercial data processing application. Mean Time To Failure (MTTR) or Mean Time To Data Loss (MTTDL) are appropriate metrics for non-stop and mission critical applications. Environmental factors, such as megabytes per square foot (or cubit foot) and megabytes per watt, are relevant for data centers whose storage demands are increasing but whose machine room space and power cannot continue to grow.

Determining the (unformatted) capacity of a disk is straightforward: vendors quote the capacity in terms of *megabytes*. The performance metric most frequently used by disk vendors is *megabytes per second*. However, more than one transfer rate can be cited: peak transfer rate from an embedded track buffer, instantaneous transfer rate from the disk head, track transfer rate (a function of sector size and influenced by the formatting efficiency of the disk), cylinder transfer rate (includes head switch times), and spiral transfer rate (includes both head switches and track-to-track seek times). The former will yield the highest rate but can be sustained only for a very short period of time. For example, a track buffer may be able to provide a transfer rate of 5 MBytes/second, but it could sustain it only for the time to empty the buffer, a few milliseconds. It takes a full rotation, over 10 ms, to refill the buffer. The track transfer rate, which takes into account the amount of space on the device allocated to intersector gaps, is a more realistic performance measure for most applications, although cylinder or spiral transfer rates are relevant for applications that access data in large chunks, such as image processing.

Note that drive manufacturers do not quote IOs/second, the most relevant metric for many data and transaction processing applications, including order entry, airline reservations, funds transfer, etc. The problem is determining just what is a "standard" I/O request, since this is so highly application dependent. It is a function of request size, the temporal and spatial locality of reference patterns (e.g., 80% of the references to a file may be to 20% of its records), and the tolerable disk service time (the time from when the application requests an I/O to the time that the requested data is delivered to the application is a function of queuing delays, and seek, rotate, and transfer times). For example, Bates [Bates89] has characterized the VAX/VMS timesharing environment as:

- Average seek distance is 10% of the full stroke of the disk
- The most frequent seek distance is 1 cylinder
- 50% of all seeks are already positioned at the correct cylinder
- Average transfer size is 4K bytes
- The most frequent transfer size is 512 bytes (one sector)
- 90% of all requests are less than or equal to 8K bytes
- 70-80% of all references are reads

Given such seek and transfer size distributions, and a target response time target (e.g., I/Os must complete within 50 ms), it is possible to obtain a realistic IOs per second metric (see Figure 2).

While raw performance is important, most users are interested in the cost to obtain a given level of performance. Users and vendors tend to think in terms of capacity cost, that is, $ per megabyte, but rarely in terms of cost per IO per second or MByte per second. For capacity sensitive environments (most timesharing environments fall into this class), $/MB is a relevant cost/performance metric. Users purchase enough disk drives to satisfy their capacity needs, rather than to satisfy a performance goal. Gray [Gray90] claims that transaction processing environ-

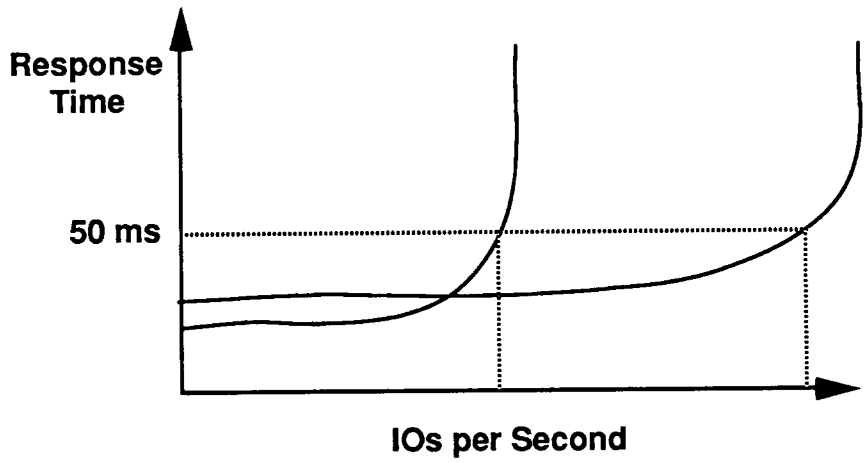

Figure 2: Two System Configurations Able to Support Different IO rates @ 50 ms Response
Response time is usually presented in terms of the maximum number of I/O requests that can be serviced within a set time. The response time curve is characterized by a long flat region that begins to climb rapidly towards infinity as the system reaches saturation.

ments are configured for cost per IOs/second. Users purchase enough disk drives (actually disk actuators) to meet their IOs per second demands. The drives are rarely filled to capacity.

To understand the differences in configuring for I/O performance and capacity for OLTP versus time sharing environments, consider the following example. Assume that either configuration is using the same building block: a 1 GByte disk drive that can service 40 random I/Os per second. Both environments need to store 10 GBytes. In addition, the OLTP performance goal is 1000 IOs/second. The most cost effective configuration for the time sharing environment is to purchase 10 disk drives and to fill them to full capacity. This yields only 400 IOs/second performance, insufficient for the OLTP environment. To achieve the latter goal, we must purchase 25 drives, leaving them 60% empty. This results in particularly poor $/MB, and illustrates the inadequacy of depending on $/MB as the storage system figure of merit for all applications.

Up to this point, we have concentrated on cost and performance. Now we turn to reliability/availability metrics. A *reliable* component is one in which there is a small probability that it will fail. An *available* system admits that components do fail, and provides mechanisms that insure that the system remains operational despite component failures, although its performance may be degraded. MTTF is a reliability metric, e.g., data cannot be accessed but is still stored on disk. MTTDL is an availability metric, e.g., the contents of a disk have been irrevocably lost.

The Mean Time To Failure curve (failure rate as a function of system lifetime) exhibits a familiar "bathtub" shape. Because of infant mortality among the system's components, the failure rate is high during its early lifetime. Then it enters a region with a constant (hopefully small) failure rate. Finally, as the system ages, it reaches the "wear out" region, with a rapidly increasing rate of failure. An artifact of the way MTTFs are computed indicates that a significant number of components will fail before the Mean Time To Failure is reached (there is a 33% probability that a drive has failed *before* its MTTF). To placate customers, many drive vendors quote a relatively low MTTF, in the 30,000 – 50,000 hour range (NOTE: 1 year = 8760 hours), assuming that customers will interpret this number as meaning that no drive will fail in that many hours (in fact, a small number of will fail regardless). A general trend in the industry is to revise these numbers upwards, into the range of 100,000 – 150,000 hours (almost 20 years!). The basis for this is not clear, since very few disks have been left powered on and in use for this length of time.

So why should a user be interested in MTTFs that are measured in decades? Under the constant failure rate assumption, the MTTF of the system scales down directly with the number of drives in the system For example, a system with a single drive may exhibit a 50,000 hour Mean Time To Failure. A system constructed from 10 such drives is expected to experience a drive failure once every 50,000/10 hours = 5000 hours, which is once every 7 months. A disk array with 10 drives including parity protection and a hot spare still has an MTTF of 5000 hours, but an MTTDL that is measured in millions of hours.

This justifies a different way of looking at the issue of availability. Suppose that a system manager has the goal of achieving a particular level of availability for his or her storage system, such as an MTTDL of one year. Figure 3 shows the effect of redundancy on the size of the storage system that meets this goal. Redundancy allows you to build a much larger storage system while still maintaining an acceptable level of availability. For storage systems constructed from many disks, a basic assumption underlying the disk array approach, some form of redundancy will be required just to achieve availabilities comparable to that of a single disk, let alone multiple disk lifetimes.

This discussion has highlighted the most obvious storage metrics, that is, performance, capacity, and availability, but this by no means completes the list. For example, other metrics may concentrate on expected uptimes (e.g., 99.95% available) or system responsiveness (e.g., 95% of all transactions complete in 2 seconds or less). These are particularly important for transaction processing environments. Also, we have not addressed the metrics related to machine room efficiency: MBytes of capacity per square foot or cubic foot, or MBytes per watt. These metrics are extremely important for applications like the Lexus/Nexus retrieval services, whose data capacity demands grow faster than their machine room space.

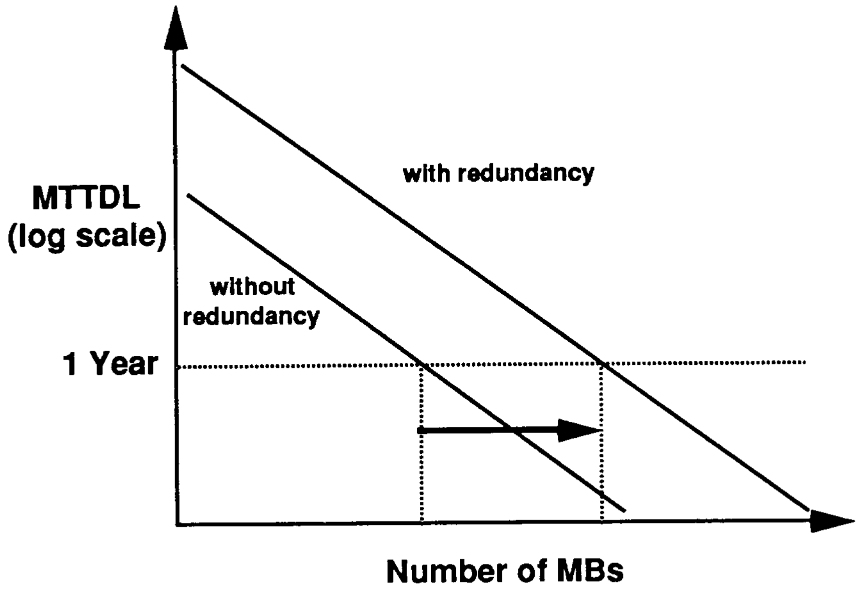

Figure 3: Ability to Support a Larger Storage System MTTDL vs. Storage System Size
A storage system architecture that yields a higher MTTDL means that the storage system can continue to grow in size while still sustaining a given level of availability.

4. A New Storage System Metric for Capacity and Performance

4.1. Mirrored vs. Arrayed Performance

In this section, we will propose a new way of looking at storage system metrics that attempts to combine cost per megabyte and cost per I/O per second. We will apply the method to compare the relative merits of mirrored disks versus parity protected disk arrays. It is important to develop such a combined metric because mirrors exhibit better cost for a given level of performance while parity arrays are more attractive based on cost for capacity. In addition, arrays can exhibit enhanced availability if they are packaged with hot spares [Gibson91].

We start with a simple model of I/O performance, based on previous work reported by Gray [Gray90]. The model computes two key elements of performance: response time and disk busy time. Response time is essentially the elapsed time to complete a request, and takes into account parallel I/O actions in the disk system. Disk busy time is related to throughput, and is the sum of the times that disks are occupied performing I/O operations. For example, a mirrored write can complete in the worse case expected time to write a single disk, since the writes to the two disks in the pair are overlapped. However, the disk busy time would record the sum of the times that the two disks are busy servicing the request.

The response time and disk busy time for mirrored read and write operations are as follows. *Seek(n)* is a function that returns the expected time to seek n heads to the same track position. Bitton [Bitton89] has shown that the expected seek time increases with the number of heads that must be simultaneously positioned.

Mirrored Response Time

Read:

$$.8 * \text{Seek}(1) + \frac{\text{Rotate}}{2} + \frac{\text{Request Size}}{\text{Transfer Rate}}$$

Write:

$$\text{Seek}(2) + \frac{\text{Rotate}}{2} + \frac{\text{Request Size}}{\text{Transfer Rate}}$$

The read response time is described as 80% of the average seek time (the 20% improvement comes from advantageous seek scheduling) plus the expected rotation time plus the transfer time. The write time is similar, except that the expected seek time has increased. Basically, we must pay a slight penalty to bring the two disk arms back to the same track position to perform the mirrored write.

Mirrored Disk Busy Time
Read:

$$.8 * \text{Seek}(1) + \frac{\text{Rotate}}{2} + \frac{\text{Request Size}}{\text{Transfer Rate}}$$

Write:

$$2 * \left[\text{Seek}(2) + \frac{\text{Rotate}}{2} + \frac{\text{Request Size}}{\text{Transfer Rate}}\right]$$

The mirrored disk busy time for read is exactly the same as the read response time, since only one disk is involved in the read operation. However, the write time is twice the write response time because both disks are kept busy servicing the I/O request.

Now we turn to N+1 parity arrays. The read and write response times are as follows:

N+1 Response Time
Read:

$$\text{Seek}(1) + \frac{\text{Rotate}}{2} + \frac{\text{Request Size}}{\text{Transfer Rate}}$$

Write:

$$\text{Seek}(2) + 1.5 * \text{Rotate} + \frac{\text{Request Size}}{\text{Transfer Rate}}$$

The read response time is similar to that of mirrored disks, but cannot take advantage of seek scheduling, since the data can only be accessed by a single actuator. For write operations, the response time formula is more complex. Two heads must be positioned, to update the data block and the parity block, and thus the penalized seek time must be used. Further, since old data and parity must be read, the bit changes determined, and new data and parity written back to disk, this will require one and one-half disk revolutions (assuming the disks are rotationally synchronized).

N+1 Disk Busy Time
Read:

$$\text{Seek}(1) + \frac{\text{Rotate}}{2} + \frac{\text{Request Size}}{\text{Transfer Rate}}$$

Write:

$$2 * \left[\text{Seek}(2) + 1.5 * \text{Rotate} + \frac{\text{Request Size}}{\text{Transfer Rate}}\right]$$

Configuration	*Response Time*		*Disk Busy Time*	
	R	**W**	**R**	**W**
N	27 ms	27 ms	27 ms	27 ms
2N	24	29	24	58
N+1	27	46	27	92

Table 1. Relative Response/Busy Times of the Three Organizations

The N+1 busy times are derived in a similar fashion as in the Mirrored case. The read busy time is the same as the response time, because only one disk is involved in the operation, and the write busy time is twice the write response, because two disks take part in the write.

To see the implications of this model, consider the following disk parameters, based on the specification of the Tandem disks described in Gray's paper:

- Seek(1) = 17 ms, Seek(2) = 19 ms
- Rotation time = 16.7 ms, Rotation/2 = 8.3 ms
- Request Size/Transfer Rate = 2 ms

Under these assumptions, the response and busy times for read and write operations are shown in Table 1. For a conventional, non-redundant "simplex" organization, the read and write response and busy times are expected to be 27 ms. In the mirrored configuration (2N), the read response and busy times are actually better, because of the advantageous seek scheduling. The write times are worse, because of the longer expected seeks associated with positioning the two heads. For the parity array (N+1), the read times are the same as a conventional disk, while the writes are considerably worse: about three times the busy time of a non-redundant disk and 60% worse than than mirrored write. While the detailed comparisons will vary with the disk parameters, the major additional expense associated with array writes versus mirrored writes is the additional rotation time (which is doubled for the disk busy time). This is an unavoidable situation given that arrays must perform a read/modify/write strategy in order to update parity.

While writes are certainly more expensive in the array organization, it is still attractive when compared with a small number of large drives. For example, consider the following comparison. A large drive has a capacity of 1.2 GBytes compared with a small drive of capacity 120 MBytes. We will construct three alternative organizations: a single non-redundant "simplex" large drive, a single pair of mirrored large disks, and a 10 + 1 + 1 (hot spare) parity array of small disks. Note that [Gibson89] shows that the spare pool can be quite small, in the range of 1% of the data disks, but we will be conservative here. The three organizations exhibit the same user available storage, 1.2 GBytes, although in the mirrored organization, the user must purchase a total of 2.4 GBytes, while in the parity array, 1.44 GBytes (120 MBytes X 12 drives) must be purchased. We assume that the system cost is determined by capacity ($/MB purchased) and that the cost per MB is the same for the small disks as the large disks (we will relax this assumption later in this section). For computing availability, we assume that the repair time for the mirrored pair is 24 hours to obtain the replacement disk and to rebuild the its contents, while it is 1 hour for the array with its hot spare.

The comparison is shown in Table 2. The relative costs are determined by the relative number of MBytes that must be purchased in the configuration. The availability is measured in "disk lifetimes", i.e., the computed mean time to failure for the collection of disks divided by the mean time to failure for a single disk. An availability of 1000 disk lifetimes for the mirrored configuration means that a mirrored pair has an expected lifetime 1000 times as long as single disk. Expressed differently, 1000 pairs would yield the same media availability as a single conventional disk. The disk array offers an intermediate level of availability, but at a more modest capacity

Configuration	*Rel. Cost*	*Avail.*	*IO/s per disk 1R:1W*	*IO/s per disk 2R:1W*	*IO/s per sys 1R:1W*	*IO/s per sys 2R:1W*
Big N	1	1	18	18	18	18
Big 2N	2	1000	12	14	24	28
Small 10+1+1	1.2	380	8	10	88	110

Table 2. Relative Cost and Performance of Large Disk versus Small Disk Configurations

cost as compared to mirroring. And while the read and write performance is degraded when compared with mirrors, the array wins this back by providing almost twice as many actuators, and a significantly better aggregate thruput.

In a bottom line comparison, the array organization costs 72% of the large mirrored configuration, provides 38% of its availability, and yields almost four times the performance. An array based on small disks is clearly superior to a mirrored configuration based on large disks.

Because of its advantage on the basis of IOs per actuator, mirroring will yield higher performance if we had compared mirrored small disks versus arrayed small disks. A major issue is how best to configure an array of small disks, as collections of mirrored pairs or as parity arrays. We tackle this question in the next subsection.

4.2. Capacity vs. Performance

Parity arrays and mirroring represent two points within a space of tradeoffs between capacity and performance. To better examine these tradeoffs, it is useful to construct a graph of MBs (capacity) versus IO/second (performance).

In Figure 4, we display two separate graphs: IOs per second as a function of cost and MB as a function of cost. In the performance case (the left chart of the figure), as we spend more money, we buy more disks and achieve a higher rate of IOs. From the previous subsection, we know that mirrored disks deliver more IOs per actuator than parity arrayed disks. Assuming that we are purchasing the same kinds of disks (thus costing the same) for either the mirrored or arrayed configuration, then the change in I/Os per second as a function of cost will be greater for the mirrored case. To reach a particular I/O performance goal, mirrored disks will attain it at

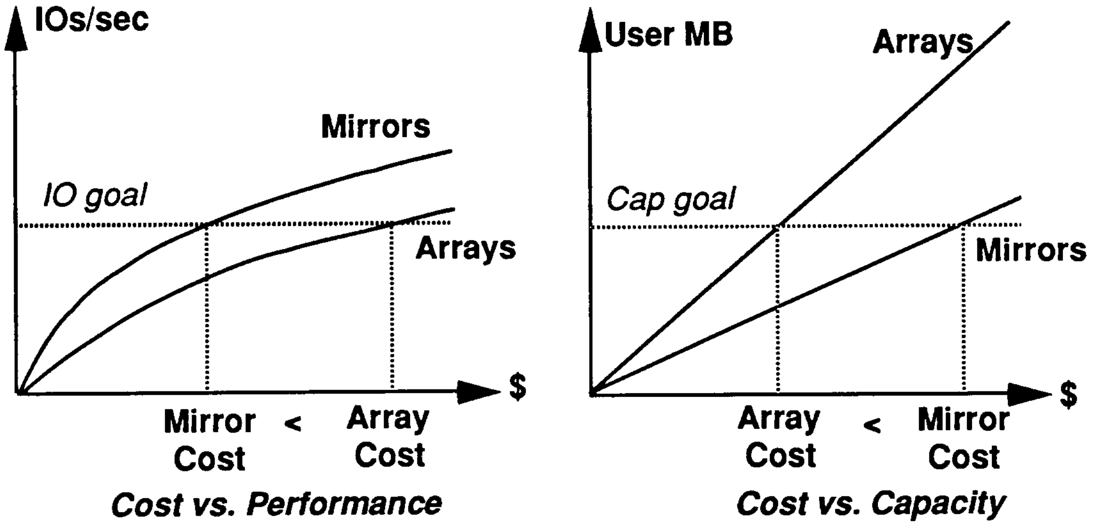

Figure 4: Cost versus Performance (IO/second) and Capacity (MB)
For a given performance goal, mirrors will achieve at a lower system cost. On the other hand, the disk array will meet a capacity goal at a lower cost. The best organization is the one that fulfills both performance and capacity goals for the lowest overall cost.

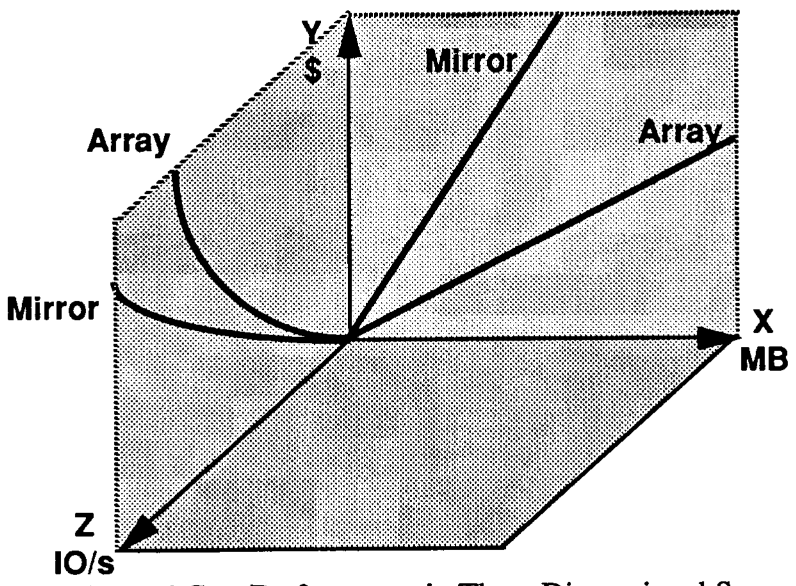

Figure 5: Cost/Capacity and Cost/Performance in Three Dimensional Space
Cost/Capacity and Cost/Performance are combined to form a third plane upon which to project equal cost system organizations with different tradeoffs between performance and capacity.

lower costs than a parity array. The graph is conceptual, and shows the diminishing performance returns as more disks are added to the configuration (e.g., controller or other system bottlenecks limit performance). In reality, the curves would look more like a stairstep, as performance improves in discrete steps as we incrementally add more disks.

The right part of Figure 4 shows a similar kind of conceptual chart, this time comparing reliable storage as a function of cost. Since the mirrored configuration must purchase more physical storage than the array method for a given amount of user storage, it comes as no surprise that the cheapest way to achieve a given capacity goal is with arrays. Once again, the chart should really be a stairstep, as storage is purchased in discrete increments.

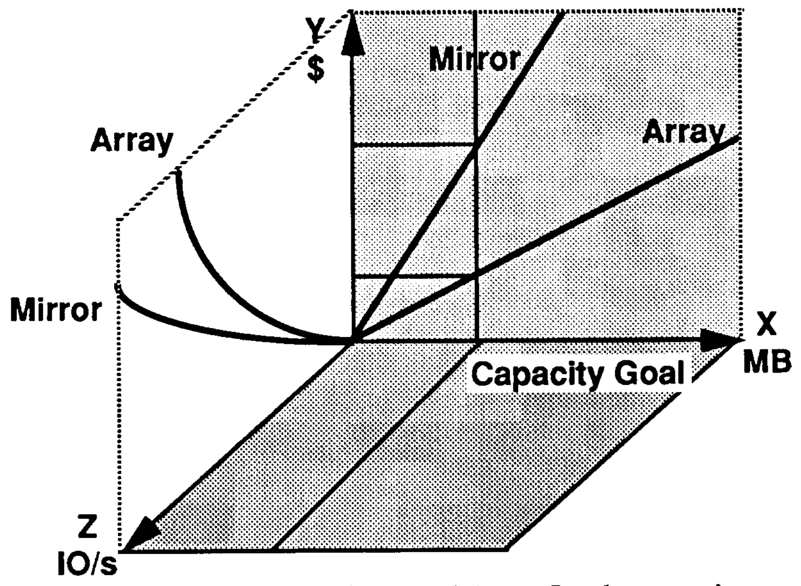

Figure 6: Capacity Goal and Costs for Mirror and Array Implementation
For a given capacity goal (e.g., user MBytes), the array organization will yield a lower overall system cost than a mirrored organization.

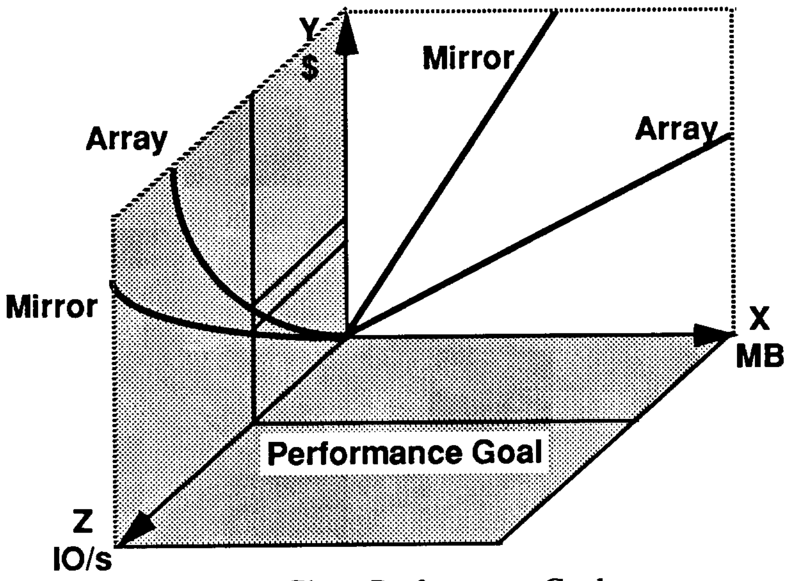

Figure 7: Relative Costs to Reach a Given Performance Goal
Similarly, the mirrored system will achieve the performance goal at lower cost than an arrayed system. This assumes that both systems are constructed from the same building block disks.

Figures 5, 6, and 7 use the charts of Figure 4 to construct a three dimensional space that trades off between cost, performance, and capacity. We choose to set the X axis to capacity, the Y to cost, and the Z to performance. The XY and YZ planes are simple transformations of the graphs of Figure 4. Their projection onto the XZ plane represents the tradeoff between capacity and performance.

Figure 6 shows the effect of setting a capacity goal for a particular configuration. The mirrored implementation costs more than the array implementation, as can be seen by the intersection with the cost axis. Figure 7 applies the same idea, except for a particular performance goal. In this case, the array implementation will cost more to attain the same level of performance.

Combining Figures 6 and 7 yields Figure 8. For each configuration, we have two cost points: the cost to achieve a given level of performance and the cost to achieve a given capacity goal. The best configuration is the one that minimizes the worst case cost. For the array, the maximum cost is attained in achieving the I/O goal. For the mirrored configuration, it is incurred for the capacity goal. If the array performance cost, C_A, is less than the mirrored capacity cost, C_M, then arrays are the preferred configuration as shown in Figure 8. Of course, with a more ambitious performance goal, mirroring may have yielded the lower maximum cost.

There are regions of the capacity versus performance plane in which arrays dominate mirrors, and vice versa. We examine this in more detail in the next two subsections.

4.3. Model Cost for a Storage Capacity Q

In this section, we will derive a more realistic model for calculating the cost of a desired storage capacity for the kinds of disk system organizations described above: simplex, mirrored, and parity array. Starting with a generic cost model:

$$\text{Cost} = K + R * D + C * Q \qquad \text{(Equation 1)}$$

where
K = fixed cost of the configuration, such as the controller and host cables
D = the number of disks required to store the data
R = costs proportional to the number of disks, such as cabinets and disk packaging

Q = number of megabytes
C = cost per megabyte

Let
C_D = cost per megabyte for a single disk
C_S = cost per megabyte for a non-redundant simplex string of disks
C_M = cost per megabyte for a mirrored disk
C_A = cost per megabyte for an N+1 array

Then

$$C_S = C_D \qquad \text{(Equations 2)}$$
$$C_M = 2 * C_D$$
$$C_A = \frac{N+1}{N} * C_D$$

With
m_D = the number of megabytes per disk

Then

$$D_S = Q/m_D \qquad \text{(Equations 3)}$$
$$D_M = 2 * Q/mD$$
$$D_A = \frac{N+1}{N} * \frac{Q}{m_D}$$

Combining equations 1, 2, and 3 yields the following equations for the configuration costs:

$$Cost_S = K_S + R_S * Q/m_D + C_D * Q$$

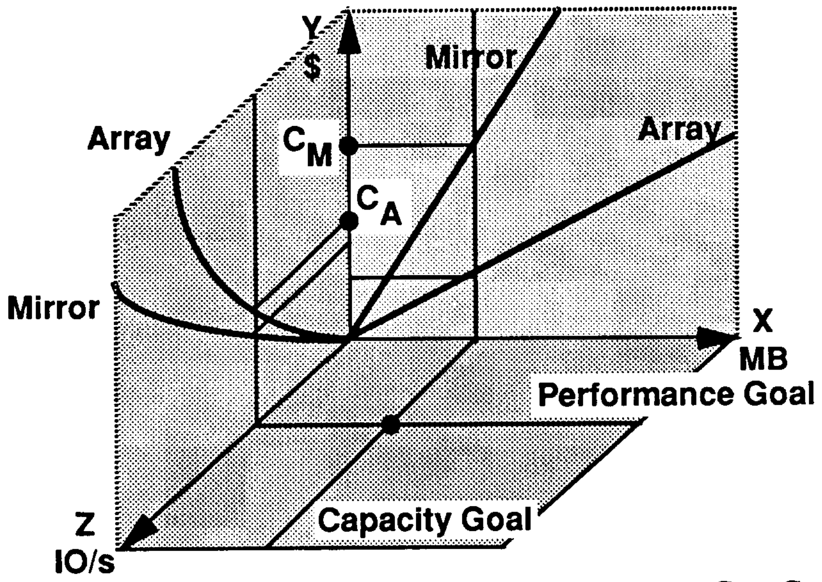

Figure 8: Using the Three Space Representation to Find the Lowest Cost Configuration
By combining the analysis of Figures 6 and 7, it is possible to determine the configuration that meets both constraints for the lowest cost. In this case, the array has obtained the lowest cost, as denoted by CA.

$\mathrm{Cost}_M = K_M + R_M * 2 * Q/m_D + 2 * C_D * Q$
$\mathrm{Cost}_A = K_A + R_A * \{[N+1]/N\} * \{Q/m_D\} + \{[N+1]/N\} * C_D * Q$

Dividing through by the desired capacity Q yields the following: (Equations 4)
$C_S = K_S/Q + R_S/m_D + C_D$
$C_M = K_M/Q + 2 * R_M/m_D + 2 * C_D$
$C_A = K_A/Q + \{[N+1]/N\} * \{R_A/m_D\} + \{[N+1]/N\} * C_D$

These final equations give a true system cost per megabyte for each of the three configurations we are considering. The equations will be useful in comparing the costs for alternative configurations with the same performance and capacity.

4.4. Model for Performance

In Section 4.1, we calculated the read and write I/O performance for the various ways to organize an array. The objective was to normalize the performance of the configuration by its capacity.

$IO_S = IO_D * D_S$

where
IO_S = I/O rate for simplex string of D_S disks.
This equation is only an approximation, since it does not take account of the diminishing increments to performance caused by adding additional disks without also increasing controller bandwidth (and hence cost).

IO_D = I/O rate of an individual disk, and

$IO_{MR} = IO_D * D_M$
$IO_{MW} = 1/2 * IO_D * D_M$
$1/IO_M = [R/IO_{MR}] + [W/IO_{MW}]$
$R + W = 1$

substituting:
$1/IO_M = [R/(IO_D * D_M)] + [W/(\{1/2\} * IO_D * D_M)]$

and simplifying:
$IO_M = IO_D * D_M/(R + 2W)$

where
IO_M = IO rate for a set of mirrored pairs (D_M disks and $D_M/2$ pairs)
This approximation does not take account of the advantage of the closest head seek scheduling algorithm, nor does it penalize writes when the heads must be resynchronized. For the example in Table 1 the reads are 13% faster than this form predicts, and the writes are 9% slower.

IO_{MR} = IO read rate for $D_M/2$ mirrored pairs
IO_{MW} = IO write rate for $D_M/2$ mirrored pairs
R,W = the proportion of reads and writes respectively in the IO stream

also

$IO_{AR} = IO_D * D_A$
$IO_{AW} = 1/4 * IO_D * D_A$
$1/IO_A = [R/IO_{AR}] + [W/IO_{AW}]$
$R + W = 1$

substituting

$1/IO_A = [R/(IO_D * D_A)] + [W/(\{1/4\} * IO_D * D_A)]$

and simplifying yields the following equation:

$IO_A = IO_D * D_A/(R + 4W)$

where

IO_A = IO rate of a parity array.

This equation does not precisely model the write penalty. It uses the approximation of 1/4 (2 reads, 2 writes for an array write). The more complete form is discussed in Section 4.1 (for the example given in table the factor would be .29).

IO_{MR} = IO read rate for a parity array of D_A disks
IO_{MW} = IO write rate for a parity array of D_A disks

Substituting Equation 3 for the I/O performance expressions derived immediately above yields the following:

$IO_S = IO_D * Q/m_D$ (Equations 5)
$IO_M = IO_D * [2 * Q/m_D] * [1/(R + 2 * W)]$
$IO_A = IO_D * \{[N + 1]/N\} * \{Q/m_D\} * \{1/(R + 4 * W)\}$

Dividing by the capacity Q yields the next set of equations:

$IO_S/Q = IO_D/m_D$ (Equations 6)
$IO_M/Q = 2 * [IO_D/m_D] * [1/(R + 2 * W)]$
$IO_A/Q = \{IO_D/m_D\} * \{[N + 1]/N\} * \{1/(R + 4 * W)\}$

In Equation 6, a reoccurring subexpression appears: the I/O rate divided by capacity. We call this parameter the *data temperature* (IO/Sec/GB). A storage system with a high data temperature supports a high I/O rate to its capacity. A solid state disk achieves a high data temperature by providing an intrinsically high I/O rate to a small capacity. A disk system achieves a high I/O rate by spreading its capacity over many disk actuators. Too much capacity under a single actuator leads to a low data temperature and inadequate performance in the I/O system.

Let T be the configuration's data temperature, that is, its sustained I/O rate over its capacity. This gives us the following equations:

$T_S = T_D$ (Equations 7)
$T_M = 2 * T_D * [1/(R + 2 * W)]$
$T_A = T_D * \{[N + 1]/N\} * \{1/(R + 4 * W)\}$

where
T_D = temperature of a single disk
T_S = temperature of a simplex string comprised of a set of disks each with temperature T_D
T_M = temperature of a mirror configuration comprised of a set of disks each with temperature T_D
T_A = temperature of an array configuration comprised of a set of disks each with temperature T_D

Consider the following hypothetical application. If it requires 10 gigabytes of data which is accessed a maximum of 400 times per second, then its data should be placed in a storage system that can support a data temperature of 40 IO/Sec/GB. Postulating a set of disks with the actuator performance described in Section 4.1, we can derive the data temperature capacities as shown in Table 3.

Capacity (GB)	Diameter (inches)	Maximum (IO/sec)	Temperature (IO/Sec/GB)	--- 1992 Cost --- ($/drive	$/MB)
.163	3.5	37	227	350	2.15
.210	3.5	37	176	400	1.90
.320	3.5	37	115	500	1.56
.420	3.5	37	88	600	1.43
.380	5.25	37	97	820	2.16
.760	5.25	37	49	950	1.25
1.100	5.25	37	34	1500	1.36
1.600	5.25	37	23	2400	1.50

Table 3. Comparison of data temperature and cost for various capacity 3.5" and 5.25" formfactor disks. Cost data (source: Dataquest) is for quantity OEM purchases.

If the application's data was stored on 7 x 1.6 GByte drives, the storage system would not be capable of providing sufficient I/O performance during peak demand periods. The data temperature of 23 IO/Sec/GByte is too low to meet the 40 IO/Sec/GByte requirement, thus the application would suffer from slow response times. The solution, assuming we are restricted to 1.6 GByte drives, would be to add four additional drives and only partially fill them with data. The eleven drives would each contain .91 GByte of data, supply 37 IO/sec, for a data temperature of 40 IO/Sec/GByte. An alternate configuration might store the data on .76 GByte drives. These are capable of supporting data temperatures of 49 IO/Sec/GByte and thus fourteen drives would contain the data and would be able to support the desired I/O rates. The deciding criteria would be cost. If eleven 1.6 GByte drives have a lower cost than fourteen .76 GByte drives, then the choice would obviously favor the first configuration. Otherwise, the configuration consisting of the .76 GByte drives would be preferred. Using the data in Table 3, eleven 1.6 GByte drives would cost $26,400 and fourteen .76 GByte drives costs $13,300. Thus, a system configured from the .76 GByte drives is preferred.

5. Application of Data Temperature Metric

The concepts and equations developed in the previous section can be used to help resolve such questions as which configuration, an N + 1 array or a 2N mirror, will give lower costs.

The first example is a simplified exercise that demonstrates the principles involved. The example compares the cost of a 10 + 1 array and a mirror that have the same data temperature capacity. The application is characterized by 70% reads and 30% writes. Now the simplification: only the cost of the storage will be considered. Controller costs, cabinet costs, etc., will be left out of the analysis:

$T_A = T_M$

Substituting from Equation 7, and dropping the appropriate terms, yields the following:

$T_d * \{[N + 1]/N\} * \{1(R + 4 * W)\} = 2 * T_D * [1/(R + 2 * W)]$

with

T_d = the temperature of the disk used in the array
T_D = the temperature of the disk used in the mirror
m_d = the capacity of the disk used in the array
m_D = the capacity of the disk used in the mirror
$IO_d = IO_D = IO$
$N = 10$
$R = .7$
$W = .3$

$T_d = IO/m_d$
$T_D = IO/m_D$

Rearranging the terms produces the following:

$m_d/m_D = \{[N+1]/[2 * N]\} * \{[R + 2 * W]/[R + 4 * W]\}$ (Equation 8)
$m_d/m_D = .38$

Constructing a 10 + 1 array from disks with half the capacity of those used in a mirrored configuration with the same capacity yields identical data temperatures, and thus equivalent performance. Table 4 show the results when Equation 8 is evaluated for various read percentages, i.e., 0<R<1.

Read %	m_d/m_D
100	.55
90	.47
80	.41
70	.38
60	.35
50	.33
40	.31
30	.30
20	.29
10	.28
0	.28

Table 4. Expansion of Equation 8 as a Function of Read Percentage

Let us now consider the costs. With the previously mentioned simplifying assumptions still in force,

$C_A = C_M$

defines a boundary between the region where the array organization has a cost advantage versus one in which mirrored organization are to be preferred. Using Equation 4, and dropping the terms for the fixed costs and variable costs that scale with the number of disks in the configuration, we obtain the following equation:

$\{[N+1]/N\} * C_d = 2 * C_D$

where

C_d = cost per megabyte for the array disk

C_D = cost per megabyte for the mirror disk

thus

$C_d/C_D = 2 * N/[N + 1]$

$C_d/C_D = 1.82$

The array will cost less if the smaller disk can be purchased for less than 1.82 times the cost per megabyte of the disk used in the mirrored configuration. If an application requires a maximum data temperature of 50 IO/Sec/GByte at 70% reads, then using the data in Table 3, a mirror configuration comprised of 1.1 GByte 5.25" diameter drives would provide a solution. A 10 + 1 array of .42 GByte 3.5" diameter drives would also be a solution. With the 1992 OEM cost per megabyte at \$1.36 for the 1.1 GByte drive and \$1.43 for the smaller .42 GByte drive, the ratio of C_d/C_D is 1.05, significantly below the boundary value of 1.82. Thus the array configuration is less expensive. Table 5 summarizes the two configurations.

	Mirror	**10 + 1 Array**
Capacity of Drive	1.1GB	.42GB
Diameter of Drive	5.25"	3.5"
Temperature of Drive	34 IO/Sec/GB	88 IO/Sec/GB
Cost/Drive	\$1500	\$600
Drive Cost/MB	\$1.36	\$1.43
# of Drives	8	11
Capacity of System	4.4GB	4.2GB
% Reads	70%	70%
Temperature of System	52 IO/Sec/GB	51 IO/Sec/GB
Cost of System	\$12.0K	\$6.6K
System Cost/MB	\$2.73	\$1.57

Table 5. A comparison of various parameters of a mirror and an array configuration each capable of supporting a data temperature of 50 IO/Sec/GB at 70% reads. Cost data (OEM) source: Dataquest.

This analysis suggests that the array configuration is a more cost effective choice for this application. We now discuss how the simplifying assumptions have effected the conclusion. Some of the terms in Equations 4 are driven by costs associated with the number of disks. However, with the mirror configuration using ten disks and the array configuration eleven, there is little difference and the effect on cost is quite small. The remaining terms are related to the fixed costs, mostly in terms of the controller. In general, a disk array controller handles data storage and retrieval as well as managing redundancy and reconstruction. On the other hand, mirroring is often implemented in the host; the actual disk controller is the same as that used for a simplex configuration. Thus, the array control cost could represent a substantial difference in system costs. In addition, disk array controllers are in their first generation while conventional disk controllers are a mature technology. This results in higher levels of integration and more fully cost reduced implementations for the latter than the former. While array controllers may be significantly more expensive than a mirrored controller today, in the long run we believe the controller costs for the two architectures will converge.

6. SUMMARY AND CONCLUSIONS

In this paper, we have presented a metric, *data temperature*, that combines aspects of capacity and performance as a metric with which to evaluate secondary storage systems. This concept has appeared from time to time in the literature, such as in [Goldstein87]. He introduces Accesses per second per GByte as a storage system figure of merit, and uses it to examine trends in disk system organizations into the mid-1990's. It is well known that the disk drive industry is driven by price per capacity, but most end users are interested in a combination of performance and capacity at a reasonable price.

The approach we have advocated uses the data temperature metric (IO/Sec/GB) to provide an easily measured parameter that can be used to characterize an application's I/O system requirements. This parameter can also be used to quantify the capability of various devices (both single disks and configurations of disks) to provide the application with the capacity and performance it demands. Since the metric is easy to measure and can be applied to both applications and devices, it serves a useful role in designing storage systems to meet specific requirements. Once again, the advantage goes to the particular configuration that satisfies the performance and capacity goals simultaneously for a lower cost.

Thus data temperature is an important storage system metric, worthy of consideration along with the classical metrics used in the design of storage subsystems:

Metric	*Units*
Cost	$
Capacity	Gigabytes
Performance	
IO rate	IO/sec
Response time	milliseconds
Transfer rate	MB/Sec
Cost/capacity	$/Megabyte
Cost/performance	$/IO/Sec
Data temperature	IO/Sec/GB
Footprint	Square feet
Power consumption	Kilowatts
Cooling requirements	BTU/Hr
Reliability	
Mean time to failure	Hours or Disk Lifetimes
Mean time to data loss	Hours or Disk Lifetimes

7. REFERENCES

[Bates90] K. Bates, "I/O Subsystem Performance," DEC Professional Magazine, "Part 1: The Four Primary Performance Measures," (April 1989); "Part 2: DSA Optimization Techniques," (August 1989); "Part 3: I/O Workloads," (October 1989); "Part 4: How to Measure and Compare Disk Performance," (March 1990).

[Bitton88] D. Bitton, J. Gray, "Disk Shadowing," Proceedings 14th Very Large Database Conference, (September 1988), pp. 331–338.

[Bitton89] D. Bitton, "Arm Scheduling in Shadowed Disks," I.E.E.E. Spring COMPCON 1989, San Francisco, (March 1990), pp. 132–136.

[Chen90] P. M. Chen, G. A. Gibson, R. H. Katz, D. A. Patterson, "An Evaluation of Redundant Arrays of Disks using an Amdahl 5890," ACM SIGMETRICS Conference, Boulder, CO, (May 1990).

[Gibson89] G. A. Gibson, "Performance and Reliability in Redundant Arrays of Inexpensive Disks," Proceedings 1989 Annual Computer Measurement Group (CMG) Conference, Reno, NV, (December 1989).

[Gibson91] G. A. Gibson, "Performance and Reliability of Disk Arrays," Ph.D. Dissertation, University of California, Berkeley, (January 1991).

[Fujitsu87] Fujitsu Product Specification, "M2361A Mini-Disk Drive Engineering Specifications (revised)," B03P-4825-0001A, (February 1987).

[Goldstein87] S. Goldstein, "Storage Performance -- An Eight Year Outlook," IBM Santa Theresa Laboratory Technical Report TR 03.308-1, (October 1987).

[Gray90] J. Gray, R. Horbst, M. Walker, "Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput", Proceedings 16th Very Large Database Conference, Brisbane, Australia, (August 1990), pp.148 –159.

[IBM87] IBM Product Specification, "IBM 3380 Direct Access Storage Introduction," IBM GC 26-4491-0, (September 1987).

[IBM89] IBM Product Specification, "IBM 0661Disk Drive Model 371,"(July 1989).

[Salem86] Salem, K., H. Garcia-Molina, "Disk Striping," *Proc. IEEE Data Engineering Conference*, Los Angeles, CA, (February 1986).

[Patterson88] D. A. Patterson, G. A. Gibson, R. H. Katz, "The Case for RAID: Redundant Arrays of Inexpensive Disks," Proceedings ACM SIGMOD Conference, Chicago, IL, (May 1988), pp. 106–113.

[Voelcker87] J. Voelcker, "Winchester Disks Reach for a Gigabyte," *IEEE Spectrum*, (February 1987), pp. 64-67.