Appeared in the Proceedings of the Hawaii International Conference on System Sciences, January 1993, pp. 40-49.

Disk Subsystem Load Balancing: Disk Striping vs. Conventional Data Placement

Gregory R. Ganger, Bruce L. Worthington, Robert Y. Hou, Yale N. Patt
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor 48109-2122
Phone: (313) 763-5396 FAX: (313) 763-4617
ganger@eecs.umich.edu

Abstract

The I/O subsystem is becoming a major bottleneck in an increasing number of computer systems. To provide improved I/O performance, as well as to accommodate growing storage requirements, disk subsystems are increasing in size. A major hurdle to obtaining the performance available from these large disk subsystems is load imbalance, or disk skew. Dynamic data placement, the conventional load balancing technique, is usually inadequate to deal with load imbalance because it is forced to accept atomic data sets with rapidly changing access patterns. We name this rapid fluctuation **floating load imbalance** *and distinguish it from the conventional view of load imbalance, referred to as fixed load imbalance. Dynamic data placement also becomes increasingly difficult as the number of disks in the subsystem grows. Disk striping at a high granularity is suggested as a solution to floating load imbalance, the atomic data set problem and the complexity of balancing large disk subsystems. Disk striping uniformly spreads data sets across the disks in the subsystem and essentially randomizes the disk accessed by each request. This randomization effectively handles both fixed and floating load imbalance. Unlike dynamic data placement, disk striping does not become more complex as the number of disks increases. While a more optimal load balance may be possible for some very well understood and well-controlled environments, disk striping should provide significantly improved load balance with reduced complexity for many applications. This improvement will result in shorter response times and higher throughput.*

1 Introduction

Processor speeds and main memory sizes have been improving at a much faster rate than disk access times. As a result, the I/O subsystem is becoming the bottleneck in an increasing number of systems. Adding disk drives to the storage subsystem is a common solution to this problem. The number of disks in storage subsystems is also growing due to rapidly increasing capacity requirements. A storage subsystem with a large number of disks can provide significant access concurrency. To realize this concurrency, as many of the disk drives as possible should be well-utilized. While part of the responsibility for achieving this goal lies with the application program, most lies with the storage manager. The largest obstacle faced by the storage manager in reaching the desired utilization is load imbalance, or disk skew [Kim86].

Generally speaking, a storage subsystem is said to suffer from load imbalance when some of the disk drives receive higher percentages of the access stream than do others. This skewed access is often referred to as the 80/20 Rule or, in more extreme cases, the 90/10 Rule [Wilm89]. The 80/20 rule suggests that twenty percent of the resources (disk drives) receive eighty percent of the accesses, while the remaining eighty percent of the resources receive only twenty percent. Furthermore, the percentages are applied recursively. For example, twenty percent of the twenty percent receive eighty percent of the eighty percent. This load imbalance severely limits the performance of the storage subsystem. If a disk drive can handle N accesses per second and the most heavily used disk drive receives P percent of the accesses, then the 80/20 rule argues that, even for large disk subsystems, the maximum throughput is 100N/P accesses per second. Even neglecting maximum throughput, response time is severely impaired under skewed access patterns because many requests wait in the access queue for the busiest disk(s) while other disks remain idle.

The 80/20 rule can also be applied to logical data chunks which are smaller than a disk. The physical placement of the data comprising these chunks is responsible for load imbalance. We designate the temperature of a chunk of data or a disk as a qualitative measure of the load which it receives. Chunks which receive considerable loads are referred to as hot.

Fixed load imbalance, which is described by the 80/20 Rule, is only the coarse grain form of load imbalance. While large performance problems can be caused by fixed load imbalance, it is much easier to identify and deal with than its counterpart, *floating load imbalance*. Floating load imbalance exists when the data chunks which are hot change rapidly over time.

The conventional method of dealing with load imbalance, dynamic data placement, is simply insufficient. It requires significant time and effort to implement and usually fails to provide a dependable solution. Increasing the number of disk drives in the storage subsystem makes it even more difficult to balance via dynamic data placement schemes. Dynamic data placement is unable to deal well with floating load imbalance and is usually constrained to placing atomic data sets. The restriction of atomic data sets can be alleviated to some degree by partitioning problem data sets. Unfortunately, this often requires significant effort on the part of the storage manager. We view data set splitting as a small step in the direction of disk striping (which splits all data sets into small pieces). The storage management effort involved with using disk striping is much less than the effort necessary for splitting data sets. Therefore, rather than taking small steps in response to drastic performance problems, we prefer to look at the performance offered by a larger step, disk striping.

Disk striping involves combining the capacities of multiple disk drives into a single logical data space. This logical data space is then spread across the physical drives, in a round robin fashion, one unit at a time. Disk striping statistically removes fixed load imbalance and reduces floating load imbalance by randomizing the disk accessed by each request. This is accomplished by spreading data among the disks using a pseudo-random function, very much like hashing. In fact, disk striping provides load balance similar to record-hashing in databases, but at a system administration level rather than inside the database and without the problem of skewed attribute values. Unlike manual data placement, an increase in the number of disks does not introduce additional complexity. While it may be possible in some extremely well understood and well controlled environments to obtain better than a random load balance, disk striping will provide a good, dependable load balance with lower complexity for many applications.

Using disk striping to balance a disk subsystem has other benefits. First, data placement can be used to reduce seek time and rotational latency without affecting the load balance. In addition to balancing the overall access stream, disk striping also balances subsystems in which some requests are more important or require more time to complete than others. by evenly distributing each class of requests among the disks. This can be important when some requests are more important or require more time to complete than others. Finally, disk striping provides a very clean, predictable method for load balancing other disk subsystem components, such as busses and controllers.

The remainder of the paper is organized as follows. Section 2 describes fixed and floating load imbalance. Section 3 discusses the conventional scheme of reducing load imbalance, namely dynamic data placement. Section 4 provides a general description of disk striping. Previous work on disk striping is also summarized briefly. Section 5 discusses the effects of disk striping on load imbalance. Section 6 describes our experimentation methodology. Section 7 presents experimental validation for our premises. Section 8 provides additional experimental results. Section 9 presents our conclusions and some topics for future research.

2 Two Types of Load Imbalance

To examine the issue of load imbalance one must have an appropriate load indicator. The number of accesses seen by a disk over a period of time is not an appropriate metric, because it does not provide information about whether resource conflicts have been a problem or not. Load balance is most closely tied to queue time which, discounting the seek reordering possible in larger queues, is the only portion of disk response time affected by load imbalance. In fact, one might consider a system to be perfectly balanced if the queue times experienced by requests at each disk at a given point in time are equal. Queue length is closely related to queue time and may be easier to measure at given points in time. [Zhou87] and [Berr91] both provide experimental evidence showing that queue length is an appropriate load indicator. Therefore, we will use queue lengths as load indicators in this paper.

We identify two types of load imbalance that a disk subsystem can experience. First, there is the type of load imbalance indicated by the 80/20 rule. This form of load imbalance is characterized by its consistency. That is, the load imbalance is always present and each disk maintains approximately the same temperature over time. We call this *fixed load imbalance*. Unfortunately, we believe that fixed load imbalance is somewhat rare in the real world. We submit that it is more common for the temperature of the disks in the subsystem to vary widely over time, and, in particular, for the disks which are hottest to vary. Because the highest temperature moves, or floats, among the disks, we call this *floating load imbalance*.

It should be understood that there is no fundamental difference between these types of load imbalance. In fact, fixed load imbalance is a degenerate case of floating load imbalance. Fixed load imbalance is simply floating load imbalance where an extremely long time passes before the temperatures of the disks change. Also, floating load imbalance affects performance in essentially the same way as fixed load imbalance, albeit to a lesser degree. Both result in large queues, but in the latter case busy disks can catch up once their hot spots float to other disks. We distinguish between the two to highlight the importance of floating load imbalance, which generally receives less attention than it merits, and to simplify the discussion of each.

Most descriptions of load imbalance are in terms of fixed load imbalance, often suggesting the existence of an 80/20-like Rule [Wilm89] [Scra83]. There are several exceptions to this, however. For example, [Buze90] suggests that load imbalance is caused by very gradual changes in access patterns. Such a situation would appear at any point in time as fixed load imbalance, but would be floating very slowly. While not discussing the workload seen by specific disks, [McNu86] shows that the amount of load imbalance, which he represents by a skew factor, changes rapidly throughout the day in many real storage subsystems. Even if much of this skew may be attributed to fixed load imbalance, the

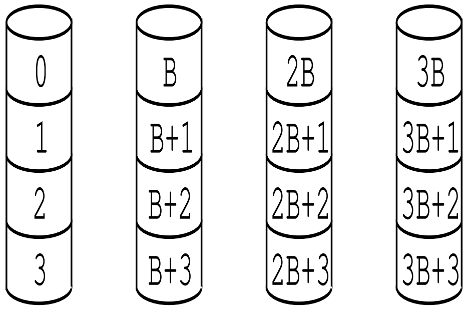

Figure 1: Conventional data placement. (B data units per disk)

rapidly changing skew factors indicate that floating load imbalance does exist in these environments. [McEl88] describes the existence of sporadically busy data sets which would be very likely to cause serious floating load imbalance.

3 Dynamic Data Placement

In a conventional disk subsystem, the data on each disk are considered independent and logically sequential. Figure 1 shows the placement of data blocks within the logical storage space in an array of four independent disks, each with a capacity of B blocks. In such systems, the most commonly used load balancing scheme is dynamic data placement. Dynamic data placement involves analyzing performance data collected during important periods of activity. Load imbalance problems are addressed by moving suspected hot spots from one disk to another. This iterative process is more an art than a science and is usually performed by system administrators or specializing consultants. These people use their experience and the available information to make an educated guess at how to rearrange the data. For example, in some cases information about the average workload seen by each disk is available. Data from very busy disks are exchanged with data from relatively idle disks [Papy88] [Wolf89] [Buze90]. More aggressive schemes will use information about individual data sets and may consider variations in the temperatures when making the placement decisions [McEl88].

There are essentially two problems with these algorithms for load balancing. First, they can be quite expensive. Performance may be degraded throughout the day due to the data collection process, considerable expert human time may be required to determine whether and how to rearrange the data, and a large number of disk accesses may be necessary to rearrange the data. This final step would need to be scheduled during a period of little or no activity to avoid serious performance degradation. The human effort described in the papers cited above is considerable, and some attempts have been made to reduce it [Wolf89] [Asch89].

The second problem is that dynamic data placement rarely succeeds in solving the load imbalance problem. There are two major problems which prevent dynamic data placement from being successful. The first problem is that conventional data placement schemes must deal with atomic data sets, data sets which are single logical units and must be placed in the storage subsystem as such. Placing a logical unit of data in a conventional storage subsystem entails placing it entirely on a single disk. This problem can prevent dynamic data placement schemes from solving even fixed load imbalance in many cases. Partitioning problem data sets can be very helpful when combating load imbalance [Papy88]. Unfortunately, this generally requires changes to tables in the software which owns the data set and may require considerable effort.

Floating load imbalance is the other major difficulty faced by dynamic data placement schemes. If access patterns changed very slowly, then it would simply be a matter of rebalancing occasionally as suggested in [Wolf89] and [Buze90]. We believe, however, that load imbalance floats very quickly (on the order of seconds and/or minutes) in many environments. Floating load imbalance makes it probable that disk temperatures present during the measurement process will not accurately predict future disk temperatures. This is similar to the effect noted by Zhou concerning CPU loads [Zhou88]. He found that CPU loads are very difficult to predict, even with current load information. Because hot spots float too frequently for the data placement to be adjusted between temperature changes, a single placement must be found to load balance for time-varying hot spots. This requirement and the atomic data set problem make it very difficult to determine a good data placement.

4 Disk Striping

Disk striping, or disk interleaving, consists of combining the data space of several disks and spreading the data across these disks such that the first stripe unit is on the first disk, the second stripe unit is on the second disk, and the Nth stripe unit is on disk (N-1 mod M)+1, where M is the number of disks involved. (See Figure 2)

Disk striping has been used by many companies, including IBM, Digital Equipment Corporation and Amdahl [Gray90], to improve data bandwidth for a given request by transferring data in parallel to or from multiple disks. In addition, there have been many studies of the performance consequences of disk striping [Kim86] [Redd89] [Chen90]. These studies have shown that there is a definite trade-off between using multiple disk drives to handle one request and using the disk drives to handle separate requests when possible. This trade-off is embodied in the choice of stripe unit. It appears that for any given workload an optimal stripe unit can be chosen. In [Chen90] algorithms for choosing the stripe unit were developed assuming various types of workload information. Chen and Patterson

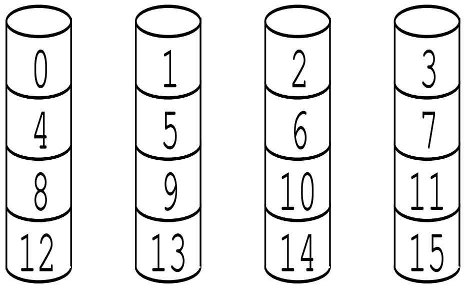

Figure 2: Striped data placement.

showed that the most important parameter in the choice of a stripe unit is the amount of access concurrency in the workload. In particular, the stripe unit should increase with the level of concurrency so that the likelihood of using more than one disk per request decreases. Except where otherwise noted, all further discussion of disk striping will be in terms of stripe units large enough to ensure that most requests use a single disk.

It has also been suggested that striping may be undesirable for transaction processing applications [Ng89] [Gray90]. Because these applications are characterized by frequent small disk accesses, it would be undesirable to use more than one disk for any single request. Therefore, one would need to stripe at a very coarse granularity. The arguments made indicate that disk striping is not worth the implementation effort for these applications.

5 Effect of Disk Striping on Load Imbalance

Disk striping using a small stripe unit, such as bit or byte, can balance the load seen by the disks in the stripe set [Kim86]. This is accomplished by simply using all disk drives for all requests, which causes each disk to see exactly the same workload.

Disk striping can also help with the load balancing problem in cases where requests are small and each is handled by a single disk. Striping breaks up hot data sets and distributes them among the disks in the subsystem. This helps reduce fixed load imbalance by increasing the number of disks which receive accesses to the hot data sets. Floating load imbalance is reduced in the same fashion because changing the hot data sets will not alter the fact that they are spread over more disks than in a non-striped storage subsystem.

Disk striping also reduces load imbalance by emulating the effect of mapping each access to the appropriate disk with a uniform random variable. When a read or write request arrives, the disk which is to be accessed is determined by a simple equation, such as ((block number DIV stripe unit) MOD number of disks). This equation has the effect, much like hashing, of uniformly randomizing the disk to be accessed. In the case of requests which require accesses to multiple disks (because of the striping), the first disk is pseudo-randomly chosen and the other disks are identified by this choice. This uniform randomization can significantly reduce load imbalance. Over a large number of requests, such a subsystem will not suffer from fixed load imbalance; the number of requests to each disk in an N disk subsystem will be close to 1/N of the total number of requests. More importantly, however, this randomness will balance the number of requests to each disk in any short period of time with high probability. This randomization occurs regardless of the location requested. Therefore, when hot spots float this randomization maintains the level of load balance, reducing the effect of floating load imbalance. Hot spots which are smaller than the stripe unit may continue to present a problem to balancing the load among the disks. This problem should be avoidable with a well-chosen stripe unit and a reasonable caching mechanism,

Disk striping may not reduce load imbalance in all cases. First, there exists the pathological case of an access pattern with a stride which is some multiple of both the stripe unit and the number of disks in the subsystem. This would certainly be the most unbalanced request stream possible with a striped subsystem, as all requests would be handled by the same disk. Due to the random nature of load balancing in a striped subsystem, there could also be times when a striped subsystem will suffer from significant load imbalance. This would occur when multiple accesses happen to map to the same disk in a short period of time. Fortunately, this phenomenon has a low probability of occurring. Because the cases where striping suffers serious load imbalance are expected to occur infrequently, disk striping should provide good average load balance.

Disk striping offers statistical balancing but is also limited to statistical balancing. Load balance, and consequently performance, could be improved by intelligent data placement strategies if the disk access patterns of the workloads of interest are *completely* understood ahead of time. Such balancing would require either that floating hot spots not exist in the workload or that it be well understood how they occur and interact so that they can be dealt with effectively. We believe that for most workloads, this is not a reasonable assumption and that the effort required for manual load balancing far outweighs the realistic potential improvement in load balance.

Some of the load balancing effects of disk striping have been noted previously. [Livn87] noted that fixed load imbalance is removed by disk striping. In [Bate91], Bates describes how disk striping could reduce load imbalance by spreading hot data sets among disks. In [Bate92], he also describes how disk striping statistically spreads requests among disks. Further, he provides a small, real world snapshot of disk striping providing a balanced load.

6 Experimental Apparatus

6.1 Traces

For this study, we have used high-resolution traces of actual disk activity from two different systems executing different applications. We feel that the diversity of our trace data lends strong support to our premises. We also believe that our results will apply to a wide variety of environments. We will present results for only one trace from each system/environment, but we have verified that the results are very similar for the other traces.

To collect these traces, we instrumented the operating system, SVR4 MP UNIX ^{TM}. This operating system is a multiprocessor version of System V Release 4 UNIX under development at NCR Columbia. Information about each disk access is gathered in real-time and stored in kernel memory. For each disk request we capture a timestamp, the disk address requested, the size of the request and the type of request (read or write). The resolution of the timestamps is better than 840 nanoseconds. The timestamps provide timing information relative to the other disk requests in the same trace but not among different traces. Our instrumentation was designed to be as unobtrusive as possible and alters performance by less than 0.01 percent.

The first system traced was an NCR 3433, a 33-MHz Intel 80486 machine. The application environment traced was the Oracle ^{TM} database system executing the TPC-B benchmark [Gray91]. In addition to the disk which contained the operating system, database object code and the other files, we used four Seagate Elite ST41600N disk drives. One of these disks was used for the log and the other three contained the remainder of the database, including data, indices and history. We scaled the database for 50 TPS and collected traces of disk activity for varying numbers of generators between eight and sixteen. Each trace captures 32K disk accesses, which represents approximately six minutes of wall clock time.

The second system traced was an NCR 3550, a symmetric multiprocessor equipped with eight Intel 80486 processors and 512 MB of main memory. The application environment was the Oracle database system executing a multiuser database benchmark based on the Wisconsin benchmark [Gray91]. This benchmark consists of a number of processes which execute database operations (joins, unions, aggregates, etc...) with short, random sleep periods between each pair of operations. In addition to the three disks containing the operating system, database object code and other files, we used fourteen HP C2247s, high-performance 3.5 inch, 1 GB disk drives. Traces were collected at various points within the execution of the benchmark. Each trace captures 1M disk accesses, which represents approximately fifty minutes of wall clock time.

6.2 Trace-driven Simulation

To experiment with many different data placements for a given access stream, we developed a disk subsystem simulator. By remapping the disk accesses, we are able to examine the consequences of various data placements. We also allow individual accesses to be decomposed into multiple requests, to study schemes such as disk striping, which may require that several disks be accessed for some requests.

To focus on the load balance of the disks containing the database, we ignore some of the requests in the traces. We ignore all log requests because their sequential nature makes a separate log disk appropriate. We also remove from consideration accesses to the hard drives, containing the operating system, database object code and other files, which account for less than 0.3 percent of the traced activity in all traces. Therefore, we simulate the database disk drives, which represents all disks accessed by the altered request stream.

Because we are only interested in the load balancing effects of various data placements, we use an average disk access time for our experiments. Using an average access time allows us to ignore data placement effects on seek time and rotational latency and consider only load balancing issues. When every disk access takes the same amount of time to complete, the only parameter which affects the response time is the queue time. In addition, because we compare the different data placements for a given stream of accesses, the only parameter which can cause a difference in the queue time is how well the requests are spread among the disks, i.e. the load balance. Therefore we use the average queue time to compare the load balance offered by various data placements for a given trace. As described in Section 2, we view queue behavior as the primary indicator of load imbalance. A value of twenty milliseconds was chosen as the access time for the Seagate disks. For the high-performance HP disks, a value of ten milliseconds was used. These times reflect the average access times in the different experimental systems.

In the next section, simulation data for each trace are presented for a number of different placement schemes. Load balance related information for the data placement used on the machine during tracing is shown. The data shown for dynamic data placement represent the best possible choice of placement. This placement assumes that the data sets must be placed as atomic units and was determined by examining all possible placements. Given this assumption, the dynamic data placement data shown are an upper bound on the load balance possible using dynamic data placement for each trace. A better load balance could undoubtedly be attained by breaking up the data sets, but this generally requires considerable effort and it is difficult to determine how to break up the data sets. In the extreme, the data sets could be broken down and placed in exactly the same way that data are placed in a striped disk subsystem. Therefore, we believe that it makes much more

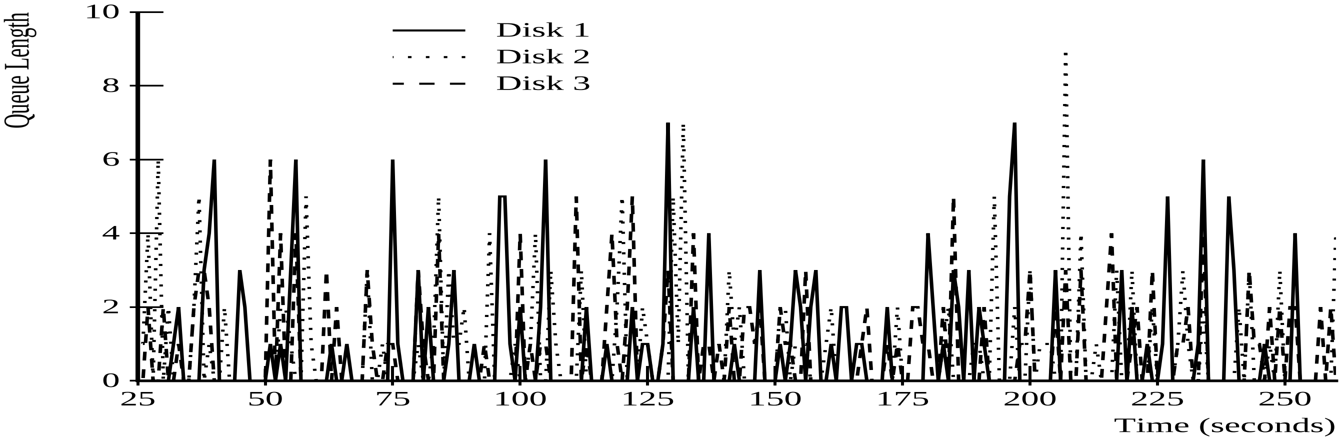

Figure 3: Sampled queue lengths from trace of TPC-B environment.

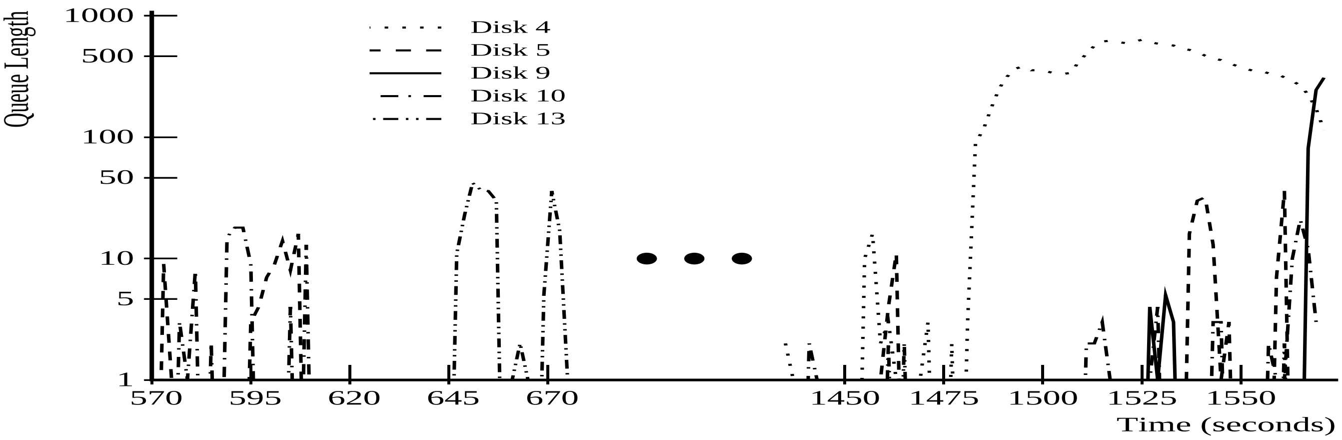

Figure 4: Sampled queue lengths from trace of database benchmark environment.

sense to simply stripe the data.

We also include results for a random load balance, which we simulate by ignoring the disk/location requested and choosing the disk to be accessed by a uniform random variable. The values provided for random load balance are the middle set of values for three simulations of random placement.

7 Validation of Premises

7.1 Floating Load Imbalance

Figure 3 shows sampled queue lengths for the TPC-B trace. The sampling interval was one second and the x-axis shows the time period of the trace used to generate the graph. The data were generated using dynamic data placement, which means that they represent activity under the best possible conventional placement. The sporadic movement of the disk queue lengths show the existence of floating load imbalance in the TPC-B environment. Not only do the access patterns change, but they do so frequently and with considerable magnitude.

Figure 4 shows the same data for the trace from the multiuser database benchmark environment. The sampling interval for this graph was two seconds. Data are shown for only five of the disks for clarity. Two different time periods are shown to highlight some particularly strong examples of floating imbalance. The queue lengths are presented in log scale to make the graph more readable. In the first period, disk 5, disk 10 and disk 13 all receive sporadic, but considerable, loads. Disk 4 and and disk 9 are completely idle. Less than thirteen minutes later, disk 10 and disk 13 exhibit much lower loads, but disk 4 and disk 9 become extremely busy. Disk 5 continues to receive a varied workload. The enormous variations in disk activity displayed in this trace (and the other traces from this environment) show that floating load imbalance is very real and very significant in this environment.

Metric	Unit	Original	Dynamic Placement	Striped (32K)	Random
Queue Length	Disk 1	0.689	0.614	0.377	0.410
	Disk 2	1.995	0.681	0.425	0.377
	Disk 3	0.171	0.622	0.408	0.425
	Total	0.951	0.639	0.404	0.404
Queue Time (ms)	Average	35.798	24.030	15.185	15.204
	% Orig	100.00	67.13	42.42	42.47
	Maximum	397.65	280.87	163.19	181.34
Number of Requests		28995	28995	28996	28995

Table 1: Comparison of placement schemes for TPC-B environment. (Access time = 20 ms)

7.2 Placement Schemes

Table 1 shows data for the TPC-B trace using a number of placement schemes. The data placement used in the system during trace collection suffers from significant load imbalance, both fixed and floating. Dynamic data placement can reduce the fixed load imbalance, but is still plagued by floating load imbalance, which accounts for most of the difference between this placement scheme and disk striping.

Disk striping provides a load balance which is almost exactly equal to random load balance. This supports the premise that disk striping provides a near-random load balance. A major reason that they are so close is the regular request sizes in the TPC-B environment. Because almost all requests are for the same small amount of data (2KB in this case), only one disk access per request is necessary (i.e., no single request requires data from multiple stripe units) when striping at a course granularity. For the experiments with this trace, a stripe unit of 32 KB was used.

While disk striping does provide a better load balance than dynamic data placement by removing the small amount of fixed load imbalance and by reducing the effects of floating load imbalance, it only improves the average response time by about 20 percent (from 44 milliseconds to 35.2 milliseconds). The main reason for this is the small size of the trace. Other traces from the same environment performed very poorly with the data placement which is best for this trace. This fact indicates that larger traces would exhibit greater differences in performance between dynamic data placement and disk striping. This premise is supported by the data from the database benchmark environment.

Data for the same experiments using a trace from the multiuser database benchmark environment are shown in table 2. The original data placement is almost as well-balanced as the best possible conventional placement (shown as dynamic placement) for this trace because an effort had been made to reduce load imbalance for the benchmark. The problem is that certain data sets receive high, but sporadic, loads. These data sets do not share disks with other data sets but still cause significant load imbalance. While the average queue lengths shown per disk would seem to indicate that fixed load imbalance is the problem, it should be noted that the high loads came at sporadic intervals for the busiest disks, which is what we define as floating load imbalance. Also, the disks which receive the largest percentage of the workload are not the same across traces, which would support our premises.

Disk striping spreads the load among the entire subsystem, reducing the average response time by more than 77 percent (from 56.3 to 12.7) for this trace. Striping does not, however, do as well as random for two reasons. First, and most important, a significant number of requests cross stripe boundaries causing an additional 77,000 disk accesses in the striped disk subsystem. These additional accesses each require the same 10 milliseconds of service time, accounting for much of the difference in queuing between random and striped placements. In addition, some sections of data smaller than a stripe unit were hot at times, causing some floating load imbalance to affect the subsystem.

To reduce the number of additional accesses in the striped subsystem, we simulated the use of a larger stripe unit. This reduces the number of additional accesses, but increases the second effect described above. The floating load imbalance suffered by the striped subsystem due to small hot spots outweighs the benefit of reducing the number of additional requests, causing the amount of queuing (represented by queue lengths and queue times) to increase. This effect is apparent in the increased values of the average queue lengths for each disk rather than in their values relative to each other. The fact that the per disk values cannot be compared to identify the existence of floating load imbalance is a major reason that it is often not found and dealt with appropriately. A full study of the effects of stripe unit choice on performance, including load balancing, is an item for future research.

8 Experimentation

8.1 Number of Disks

The ability of disk striping to reduce floating load imbalance should not decrease as the size of the disk subsystem increases, as is presently occurring in the industry. To test this, we combined traces and increased the size of the subsystem. For the conventional disk subsystem, the accesses from each trace were issued to a separate set of disks, which emulates the effect of multiple databases working in parallel. For the striped disk subsystem, the data of the traces were combined and striped over all of the disks. Requests were spread among all disks for random data placement as well. Figure 5 shows the results

Metric	Unit	Original	Dynamic Placement	Striped (32K)	Random	Striped (64K)
Queue Length	Disk 1	0.012	0.008	0.069	0.036	0.081
	Disk 2	0.000	0.000	0.069	0.036	0.079
	Disk 3	0.085	0.085	0.070	0.037	0.078
	Disk 4	11.999	11.999	0.070	0.036	0.077
	Disk 5	0.423	0.417	0.071	0.036	0.080
	Disk 6	0.000	0.000	0.069	0.036	0.081
	Disk 7	0.003	0.003	0.069	0.036	0.076
	Disk 8	0.080	0.080	0.069	0.036	0.077
	Disk 9	1.836	1.836	0.068	0.035	0.076
	Disk 10	0.497	0.497	0.069	0.036	0.080
	Disk 11	0.000	0.000	0.069	0.036	0.079
	Disk 12	0.590	0.590	0.070	0.035	0.078
	Disk 13	0.432	0.432	0.070	0.036	0.080
	Disk 14	0.038	0.000	0.069	0.036	0.080
	Total	1.140	1.139	0.069	0.036	0.079
Queue Time (ms)	Average	46.338	46.309	2.652	1.476	3.117
	% Orig	100.00	99.94	5.72	3.19	6.73
	Maximum	6405.85	6405.85	101.44	56.89	215.28
Number of Requests		1045846	1045846	1122893	1045846	1084358

Table 2: Comparison of placement schemes for database benchmark environment. (Access time = 10 ms)

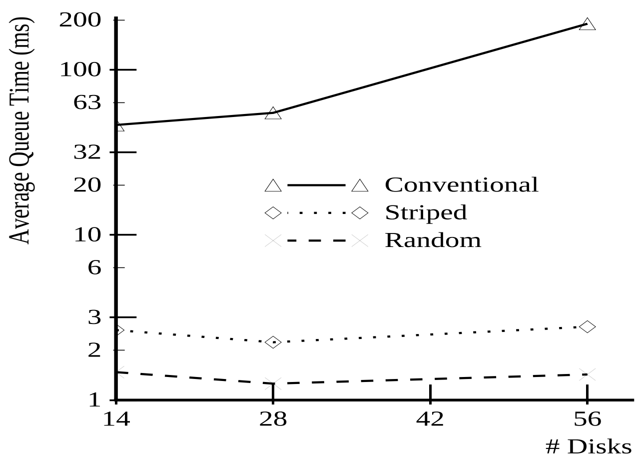

Figure 5: Data placement performance as array size increases.

for traces of the multiuser database benchmark. The results were similar for the other environments. The average queue time of the conventional subsystem for these combined traces is simply the average of the queue times for the individual traces, due to the lack of interaction between them. We see that striping does better relative to conventional data placement for the combined traces than it did for the individual trace. We attribute this to the ability of striped placement to use the greater number of disks to handle bursts of requests. The most important point, however, is that striping maintains its load balance as the size of the subsystem increases.

Adding to the number of disk drives in a conventional disk subsystem makes load balancing via dynamic data placement more difficult. Striping, on the other hand, does not increase in complexity and continues to provide a good average-case load balance.

8.2 Data Placement

We have found, as did [Livn87], that the load balance between disks provided by disk striping is independent of the placement of data within the striped disk space. This is an important result because it indicates that data placement can be used to reduce other components of the response time, such as seek time or rotational latency, without affecting the load balance.

8.3 Different Request Types

We believe that striping balances each type of request as well as the overall access stream. Table 3 provides data related to the balancing of read and write accesses for the multiuser database benchmark trace. Data are shown only for those disks having the high or low value for a column. These data show that read requests (and write requests) are balanced among the disks in the subsystem. This fact supports the more general claim that disk striping will balance different types of requests. This can be very important in environments where different types of accesses are

Disk	# of Reads	Read Qlen	# of Writes	Write Qlen
1	78542	0.067408	1315	0.001417
4	79056	0.068478	1329	0.001447
5	78960	0.069636	1327	0.001483
8	78782	0.067768	1300	0.001314
9	78885	0.066020	1307	0.001324
14	78743	0.067615	1353	0.001436

Table 3: Read/write data for the database benchmark environment.

more important or require more time to complete than others. For example, in some disk subsystems, write requests take longer to handle than read requests. This would be the case for parity-protected disk arrays (such as RAID 5) which require two accesses to the data block to complete a write. Also, many disk drives have longer seek times for writes than for reads, because the head must settle more precisely before writing. It would be appropriate to balance reads, writes and total accesses to each disk when balancing a disk subsystem having either of these characteristics.

8.4 Balancing Other Components

Disk striping also provides a clean, predictable method of balancing the workload for disk subsystem components other than disks, such as busses and controllers. Components can be configured to have equal workloads by simply connecting the same number of disks to each. This is true because the load is balanced among each of the disks in the subsystem. The load for each component can be adjusted to a desired level relative to other components by simply changing the number of disks connected to it. This further simplifies the process of managing a large disk subsystem.

9 Conclusions and Future Work

The number of disk drives in storage subsystems is growing. A significant impediment to achieving the performance offered by these disks is load imbalance. We show that floating load imbalance, the rapid changing of the temperature of each disk relative to the others, is a very real and significant problem. Balancing a subsystem by manual placement can be an extremely difficult task and will become more difficult as the number of disks increases. Further, we show that dynamic data placement fails to deal well with atomic data sets and floating load imbalance. Conversely, disk striping at a coarse granularity provides a comparatively simple data placement algorithm which provides significant improvements in load balance for many applications.

By breaking up hot data sets and randomizing the disk which is accessed for any given request, disk striping emulates a random load balance. A random load balance may not be the best possible for some extremely well understood and well controlled environments, but it will be very difficult in many cases to do better than random load balance. Disk striping does not become more difficult as the number of disks increases, unlike manual data placement. Moreover, it continues to provide a good average-case load balance as the disk subsystem grows.

We have also described other characteristics and benefits of load balancing via disk striping. The placement of data within the striped disk space does not affect the random-like load balance. Therefore, data placement can be used to reduce seek times and rotational latencies. In addition to balancing the overall access stream among the disks, striping balances each type of request. This can be important in disk subsystems which have request types which are more important or require more time to complete than others. Finally, we have explained how disk striping provides a simple, predictable method of balancing the load among other disk subsystem components, such as busses and controllers.

Further work is required for a full understanding of the issues involved with disk striping, both for load balancing and in general. The effect of the stripe unit on load balancing needs to be determined and included in algorithms for choosing a stripe unit. In this paper, we have considered only the effect of striping on load balance. Data placement, however, affects seek time and rotational latency as well as load balance. It has yet to be determined what effects striping has on these components of the access time. Also, striping effectively load balances only among disk drives with identical characteristics. If some disks are faster than others, then a proper load balance would consist of the faster disks receiving more requests than the slower disks. It may be the case that it is not a good idea to stripe over disk drives with different characteristics.

Reliability is an important issue in storage subsystems and is of growing concern as the sizes of these subsystems increase. Much work has been done to determine the cost/performance trade-offs of various redundant architectures, such as mirroring (RAID 1) and parity striping (RAID 5) [Patt88] [Chen90a] [Meno92]. We believe that the choices of data placement policy and redundancy scheme are logically independent. It has yet to be determined exactly how these choices interact in terms of performance.

It may not be appropriate to stripe over all of the disks in a very large subsystem. An obvious alternative is to break the disk subsystem into groups, or sets, of disks and stripe data over the disks in each set. Multiple sets may be desirable for systems which contain multiple disk types, data with different optimal stripe units, and/or data with different optimal redundancy schemes. The appropriate size of such sets has not been adequately explored. An important point is that load balancing between stripe sets could then become an issue. However, manual data placement may be adequate for dealing with this problem.

10 Acknowledgements

This work was partially funded by NCR Corporation, E&M - Columbia. We gratefully acknowledge their support. This paper is one result of studies of I/O subsystems being performed jointly by researchers at NCR and the University of Michigan. The project originated from conversations with Jimmy Pike and Mark Campbell, and is being followed through by Roy Clark, all of NCR. Their interest and continuing support is greatly appreciated. We also wish to thank Richie Lary of Digital Equipment Corporation and the other members of our research group at the University of Michigan for technical discussions on the various I/O issues.

Finally, our research group is very fortunate to have the financial and technical support of several industrial partners. We are pleased to acknowledge them. They include NCR, Intel, Motorola, Scientific and Engineering Software, HaL, and Hewlett-Packard.

References

[Asch89] J. Aschoff, "Performance Management of Storage Subsystems", *CMG Proceedings*, 1989, pp. 730-739.

[Bate91] K. Bates, "I/O Subsystem Performance", *DEC Professional*, November 1991, pp. 60-70.

[Bate92] K. Bates, "I/O Subsystem Performance", *DEC Professional*, February 1992, pp. 42-49.

[Berr91] R. Berry, J. Hellerstain, J. Kolb, P. VanLeer, "Choosing a Service Level Indicator: Why Not Queue Length?", *CMG Proceedings*, 1991, pp. 404-413.

[Buze90] J. Buzen, A. Shum, "I/O Performance Trade-Offs and MVS/ESA Considerations", *CMG Proceedings*, 1990, pp. 695-702.

[Chen90] P. Chen, D. Patterson, "Maximizing throughput in a striped disk array", *Proceedings of the 17th International Symposium on Computer Architecture*, 1990, pp. 322-331.

[Chen90a] P. Chen, G. Gibson, R. Katz, D. Patterson, "An Evaluation of Redundant Arrays of Disks using an Amdahl 5890", *Proceedings of SIGMETRICS*, 1990, pp. 74-85.

[Gray90] J. Gray, B. Horst, M. Walker, "Parity Striping of Disk Arrays: Low-Cost Reliable Storage with Acceptable Throughput", *Proceedings of the 16th VLDB Conference*, August 1990, pp. 148-161.

[Gray91] ed. J. Gray, *The Benchmark Handbook for Database and Transaction Processing Systems*, 1991.

[Kim86] M. Kim, "Synchronized Disk Interleaving", *IEEE Transactions on Computers*, November 1986, pp. 978-988.

[Livn87] M. Livny, S. Khoshafian, H. Boral, "Multi-Disk Management Algorithms", *SIGMETRICS*, 1987, pp. 69-77.

[Majo87] J. Major, "Empirical Models of DASD Response Time", *CMG Proceedings*, 1987, pp. 390-398.

[McEl88] M. McElwee, "The Real World of DASD Management", *CMG Proceedings*, 1988, pp. 672-677.

[McNu86] B. McNutt, "An Empirical Study of Variations in DASD Volume Activity", *CMG Proceedings*, 1986, pp. 274-283.

[Meno92] J. Menon, J. Kasson, "Methods for Improved Update Performance of Disk Arrays", *Proceedings of the Hawaii International Conference on System Sciences*, 1992, pp. 74-83.

[Ng89] S. Ng, "Some Design Issues of Disk Arrays", *COMPCON*, Spring 1989, pp. 137-142.

[Papy88] W. Papy, "DASD I/O Performance Tuning: A Case Study of Techniques and Results", *CMG Proceedings*, 1988, pp. 665-671.

[Patt88] D. Patterson, G. Gibson, R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", *ACM SIGMOD*, May 1988, pp. 109-116.

[Redd89] A.L.N. Reddy, P. Banerjee, "An Evaluation of Multiple-Disk I/O Systems", *IEEE Transactions on Computers*, December 1989, pp. 1680-1690.

[Scra83] R. A. Scranton, D. A. Thompson, D. W. Hunter, "The Access Time Myth", IBM Research Report, RC 10197, September 21, 1983.

[Wilm89] R. Wilmot, "File Usage Patterns from SMF Data: Highly Skewed Usage", *Computer Measurement Group*, 1989.

[Wolf89] J. Wolf, "The Placement Optimization Program: A Practical Solution to the Disk File Assignment Problem", *SIGMETRICS Proceedings*, 1989, pp. 1-10.

[Zhou87] S. Zhou, "An Experimental Assessment of Resource Queue Lengths as Load Indices", *Winter USENIX Proceedings*, 1987, pp. 73-82.

[Zhou88] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing", *IEEE Transactions on Software Engineering*, Volume 14, No. 9, September 1988, pp. 1327-1341.