Evaluation of Object Placement Techniques in a Policy-Managed Storage System

Pawan Goyal† **Peter Radkov** and **Prashant Shenoy**

†Storage Systems Department,
IBM Almaden Research Center
San Jose, CA 95120
goyalp@us.ibm.com

Department of Computer Science,
University of Massachusetts
Amherst, MA 01003
{pradkov,shenoy}@cs.umass.edu

Abstract

Storage management cost is a significant fraction of the total cost of ownership of large, enterprise storage systems. Consequently, software automation of common storage management tasks so as to reduce the total cost of ownership is an active area of research. In this paper, we consider a policy-managed storage system—a system that automates various management tasks—and focus on the problem of the storage allocation techniques that should be employed in such a system. We study two fundamentally different storage allocation techniques for policy-managed systems: narrow and wide striping. Whereas wide striping techniques need very little workload information for making placement decisions, narrow striping techniques employ detailed information about the workload to optimize the placement and can achieve better performance. We systematically evaluate this trade-off between "information" and performance using a combination of simulations and experiments on a storage system testbed. Our results show that an idealized narrow striped system can outperform a comparable wide-striped system at low levels of utilization and for small requests. However, wide striping outperforms narrow striped systems when the system utilization is high or in the presence of workload skews that occur in real I/O workloads. Our experiments show that the additional workload information needed by narrow placement techniques may not necessarily translate to better performance. We identify the stripe unit size to be a critical parameter in the performance of wide striped systems, and based on our experimental results, recommend that (i) policy-managed systems use wide striping for object placement, and (ii) sufficient information be specified at storage allocation time to enable appropriate selection of the stripe unit size.

1 Introduction

Enterprise storage systems are widely used to meet the online storage needs of modern organizations. An enterprise storage system consists of a large number of storage elements, such as RAID arrays, that provide online storage to applications such as databases and networked file systems. Managing a large storage system can be a complex task, especially since applications that access these systems have diverse performance needs. Studies have shown that management cost is a significant fraction (75-90%) of the total cost of ownership of a storage system [2, 15]. Storage management consists of tasks such as allocation of storage to applications, backup and disaster recovery, performance tuning, and fault diagnosis and recovery. While some tasks such as fault diagnosis are complex and require human involvement, others such as storage allocation, backup, and failure recovery are amenable to software automation. Consequently, the design of techniques to automate storage management tasks so as to reduce the cost of ownership is an active area of systems research.

We are building a *policy-managed storage system* to automate storage management. In a policy managed system, the management objectives are encoded in a set of policies; specific actions to meet these objectives are then determined and executed automatically by the system. For example, the backup and recovery policy may specify that a storage object should be recoverable to a state that is at most five minutes old in the event of a disaster. Similarly, requests for allocating storage may specify desired performance constraints on the throughput and the response times. In both cases, the actions necessary to meet the specified requirements are determined and executed automatically; such automation can significantly reduce the cost and complexity of storage management. In this paper, we focus on the automation of tasks that govern system performance; automation of other tasks such as backup and recovery is well understood and has been implemented in commercial products.

The primary research challenge in the design of a policy-managed storage system is to ensure that the system provides performance that is comparable to a human-managed system, but at a lower cost. Consequently, the storage allocation algorithms that determine object placement, and thus the performance, are crucial to the success of a policy-managed storage system. Object placement techniques for large storage systems have been extensively studied in the last decade, most notably in the context of disks arrays such as RAID [4, 8, 9, 10, 21]. Most of these approaches are based on *striping*—a technique that interleaves the placement of objects onto disks—and can be classified into two fundamentally different categories.

Techniques in the first category require a priori knowledge of the workload and use either analytical or empirically derived models to determine an optimal placement of objects onto the storage system [4, 8, 21]. An optimal placement is one that balances the load across disks, minimizes the response time of individual requests and maximizes the throughput of the system. Since requests accessing independent stores can interfere with one another, these placement techniques often employ narrow striping—where each object is interleaved across a subset of the disks in the storage system—to minimize such interference and provide isolation. In contrast, techniques in the second category assume that detailed knowledge of the workload is difficult to obtain a priori and advocate wide striping—where all objects are interleaved across all disks in the storage system. The premise behind these techniques is that storage workloads vary at multiple time-scales and often in an unpredictable fashion, making the task of characterizing these workloads complex. In the absence of precise knowledge, striping all objects across all disks yields good load balancing properties. A potential limitation though is the interference between independent requests that access the same set of disks. At least one recent study has claimed that the impact of such interference is small in practice, and that the performance of wide striping is comparable to narrow striping under many scenarios [1, 17]. Wide striping is used as a rule of thumb by an increasing number of practitioners—storage and database administrators [17]—despite claims in the research literature advocating narrow striping.

From the perspective of a policy-managed system, these two techniques have fundamentally different implications. A policy-managed system that employs narrow striping will require each allocation request to specify detailed workload parameters so that the system can determine an optimal placement for the allocated store. In contrast, systems employing wide striping will require little, if any, knowledge about the workload for making storage allocation decisions. Thus, wide striping may be easier to use from the perspective of storage applications, while narrow striped systems may make potentially better storage decisions and yield better performance. Although placement of objects in large storage systems have been extensively studied [3, 4, 5, 21], surprisingly, no systematic study of these tradeoffs of wide and narrow striping exists in the literature. This is the focus of the present work. Our work seeks to address the following questions:

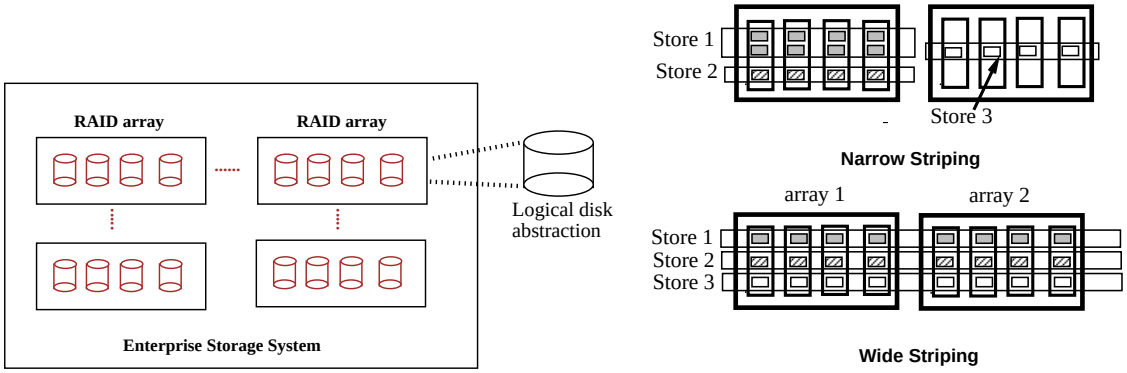
- How much (and what) information about the workload is necessary for a policy-managed system to make storage allocation decisions that provide performance comparable to manually-tuned storage systems?
- Is narrow or wide striping better suited for policy-managed storage systems? Specifically, does the additional workload information required by narrow striping translate into significant performance gains? That is, what is the tradeoff between "information" and performance?
- From a performance standpoint, how do narrow and wide striping compare against one another? What is the performance implication of the interference between requests accessing the same set of disks in wide striping? Similarly, what are the performance implications of imprecise workload knowledge and the resulting load skews in narrow striping?

We answer these questions using trace-driven and synthetic workload simulations as well as experiments on a storage system testbed. We find that an idealized narrow striped system can outperform a comparable wide-striped system at low levels of utilization and for small requests. However, wide striping outperforms narrow striped systems when the system utilization is high or in the presence of workload skews that occur in real I/O workloads. Our experiments show that the additional workload information needed by narrow placement techniques may not necessarily translate to better performance. Consequently, we advocate the use of narrow striping only when (i) the workload can be characterized precisely a priori, (ii) the system can be sufficiently provisioned to keep the utilization low, and (iii) it is feasible to use data migration to handle workload skews and workload interference. In general, we recommend that policy-managed systems use wide striping for object placement. We identify the stripe unit size to be a critical parameter in the performance of wide striped systems, and recommend that sufficient information be specified at storage allocation time to enable appropriate selection of the stripe unit size.

The rest of the paper is organized as follows. Section 2 formulates the problem addressed in this paper and presents some relevant background. Our experimental methodology and experimental results are presented in Section 3. Section 4 presents related work. Finally, Section 5 summarizes the results of the paper.

2 Background and Problem Description

An enterprise storage system consists of a large number of disks that are organized into disk arrays; a typical enterprise storage system will consist of several disk arrays. A disk array is essentially a collection of physical disks that presents an abstraction of a single large logical disk to the rest of the system (see Figure 1(a)). Disk arrays map objects onto disks by interleaving data from each object (e.g., a file) onto successive disks in a round-robin manner—a process referred to as *striping*. The unit of interleaving, referred to as a *stripe unit*, denotes the maximum amount of logically contiguous data stored on a single disk; the number of disks across which each data object is striped is referred to as its *stripe width*. As a result of striping, each read or write request potentially accesses multiple disks, which enables applications to extract I/O parallelism across disks, and to an extent, prevents hot spots by dispersing the application load across multiple disks. Disk arrays can also provide fault tolerance by guarding against data loss due to a disk failure. Depending on the the exact fault tolerance technique employed, disk arrays are classified into different RAID levels [19]. A RAID level 0 (or simply, RAID-0) array is non-redundant and can not tolerate disk failures; it does, however, employ striping to enhance I/O throughput. A RAID-1 array employs mirroring, where data on a disk is replicated on another disk for fault-tolerance purposes. A RAID-1+0 array combines mirroring with



(a) An enterprise storage system (b) Narrow and wide striping

Figure 1: An architecture of an enterprise storage system and object placement using narrow and wide striping in such systems.

striping, essentially by mirroring an entire RAID-0 array. A RAID-5 array uses parity blocks for redundancy—each parity block guards a certain number of data blocks—and distributes parity blocks across disks in the array.

With the above background, let us now consider the design of a policy-managed storage system. Consider a storage system that consists of a certain number of RAID arrays. In general, RAID arrays in large storage systems may be heterogeneous—they may consist of different number of disks and may be configured using different RAID levels. For simplicity, in this paper, we will assume that all arrays in the system are homogeneous. The primary task of the policy manager in such systems is to allocate storage to applications such that their performance needs are met. Thus, we assume that storage applications make allocation requests to the policy manager, which in turn allocates storage on one or more arrays. For example, a database application may request 15GB of storage space optimized for I/O throughput; similarly, a file system may request a certain amount of storage space and specify a certain response time requirement. We will refer to the storage allocated in response to such requests as a *store* [3]; the data on a store is collectively referred to as a *data object* (e.g., a database, a file system). The sequence of requests accessing a store is referred to as a *request stream*. Thus, we are concerned with the storage allocation problem at the granularity of stores and data objects; we are less concerned about how each application manages its allocated store to map individual data items such as files and database tables to disks.

A policy manger needs to make two decisions when allocating a store to a data object: (1) *RAID level selection:* The RAID level chosen for the store depends on the fault-tolerance needs of the application and the workload characteristics. From the workload perspective, RAID-1+0 (mirroring combined with striping) may be appropriate for workloads with small writes, while RAID-5 is appropriate for workloads with large writes.[1] (2) *Mapping of stores onto arrays:* The policy manager can map each store onto one or more disk arrays. If narrow striping is employed, each store is mapped onto a single array (and the data object is striped across disks in that array). Alternatively, the policy manager may construct a store by logically concatenating storage from multiple disk arrays and stripe the object across these arrays (a logical volume manager can be used to construct such a store). In the extreme case where wide striping is used, each store spans *all* arrays in the system and the corresponding data object is striped across all arrays (Figure 1(b) pictorially depicts narrow and wide striping). Since the RAID-level section problem has been studied in the literature [6, 23], in this paper, we will focus only on the problem of mapping stores onto arrays.

[1]Small writes in RAID-5 require a read-modify-write process, making them inefficient. In contrast, large (full stripe) writes are efficient since no reads are necessary prior to a write.

Minimum Seek	0.6 ms
Average Seek	4.7 ms
Maximum Seek	11.0 ms
Rotational Latency	5.98 ms
Rotational speed	10,000 RPM
Maximum Transfer Rate	39.16 MB/s

Table 1: Characteristics of the Fujitsu Disk

The choice of narrow or wide striping for mapping stores onto arrays results in different tradeoffs for the policy-managed system. Wide striping can result in interference when streams accessing different stores have correlated access patterns. Such interference occurs when a request arrives at the disk array and sees requests accessing other stores queued up at the array; this increases queuing delays and can affect store throughput. Observe that, such interference is possible even in narrow striping when multiple stores are mapped onto a single array. However, the policy manager can reduce the impact of such interference by mapping stores with anti-correlated access patterns on to a single array. The effectiveness of such optimizations depend on the degree to which the workload can be characterized precisely at storage allocation time, and the degree to which request streams are actually anti-correlated. No such optimizations can, however, be performed in wide striping, since all stores are mapped onto all arrays. An orthogonal issue is the inability of wide striping to exploit the sequentiality of I/O requests. In wide striping, sequential requests from an application get mapped to data blocks on consecutive arrays. Consequently, sequentiality at the application level is not preserved at the storage system level. In contrast, large sequential accesses in narrow striped systems result in sequential block accesses at the disk level, enabling these arrays to reduce disk overhead and improve throughput.

Despite the above advantages, a potential limitation of narrow striping is its susceptibility to load imbalances. Recall that, narrow striping requires a priori information about the application workload to map stores onto arrays such that the arrays are load-balanced. In the event that the actual workload deviates from the expected workload, load imbalances will result in the system. Such load skews may require reorganization of data across array to re-balance the load, which can be expensive. In contrast, wide striping is more resilient to load imbalances, since all stores are striped across all arrays, causing load increases to be dispersed across arrays in the system.

Finally, narrow and wide striping require varying amounts of information to be specified at storage allocation time. In particular, narrow striping requires detailed workload information for load balancing purposes and to minimize interference from overlapping request streams. In contrast, wide striping requires only minimal workload information to determine parameters such as the stripe unit size and the RAID level.

The objective of our study is to quantify the above tradeoff between "information" and performance and determine the suitability of narrow and wide striping for policy-managed systems.

3 Experimental Evaluation

We evaluate the tradeoffs of narrow and wide striping using a combination of simulations and experiments from a testbed storage system. Our storage system simulator simulates a system with multiple RAID-5 arrays; each RAID-5 array is assumed to consist of five disks (four disks and a parity disk, referred to as a 4+p configuration). The data

layout in RAID-5 arrays is left-symmetric. Each disk in the system is modeled as a 18 GB Fujitsu MAJ3182MC disk; the characteristics of this disk are shown in Table 1. Our storage tesbed consists of four RAID-5 arrays, each in a 7+p configuration. Each disk in our tesbed is a 9.1 GB 10,000 RPM disk with characteristics similar to the Fujitsu disk assumed in our simulations.

Depending on whether narrow or wide striping is used, each object (and the corresponding store) is placed on a single array or striped across all arrays in the system. We assume that all data blocks from an object that reside on a disk are stored contiguously (since each store is allocated a contiguous amount of space on a disk). Each data object in the system is accessed by a request stream; a request stream is essentially an aggregation of requests sent by different applications to the same store. For example, a request stream for a file system is the aggregation of requests sent by various applications accessing files in the file system. We use a combination of synthetic and trace-driven workloads to generate request streams in our simulations; the characteristics of these workloads is described in the next section. Assuming the above system model, we first present our experimental methodology and then our results.

3.1 Experimental Methodology

Recall that, narrow striping algorithms optimize storage system throughput by (i) collocating objects that are not accessed together, (i.e., collocating objects with low or negative access correlation so as to reduce interference), and (ii) balancing the load on various arrays. The actual system performance depends on the degree to which the system can exploit each dimension. Consequently, we compare narrow and wide striping by systematically varying the interference between request streams and the load imbalance. Our baseline scenario compares a perfect narrow striped system with no interference and no load imbalance with the corresponding wide striped system. Second, we compare a narrow striped system that is load balanced but has varying degrees of inter-stream interference with the corresponding wide striped system. Third, we compare a narrow striped system with load skews (and possibly inter-stream interference) to the corresponding wide striped system. We now describe each scenario in detail.

Our baseline experiment compares a perfect narrow striped system with the corresponding wide striped system. In case of narrow striping, we assume that all arrays are load balanced (have the same average load) and that there is no interference between streams accessing an array. However, these streams will interfere when wide striped and our experiment quantifies the resulting performance degradation. Observe that, the difference between narrow and wide striping in this case represents the *upper bound* on the performance gains that can be accrued due to intelligent narrow striping. Our experiment also quantifies how this bound varies with system parameters such as request rates, request size, system size, stripe unit size, and the fraction of read and write requests.

Next, we compare a narrow striped system with varying degrees of interference to a wide striped system with the same workload. To introduce interference in narrow striping, we assume that each array stores two independent objects. We keep the arrays load balanced and vary the degree of correlation between streams accessing the two objects (and thereby introduce varying amounts of interference). We compare this system to a wide striped system that sees an identical workload. The objective of our experiment is to quantify the performance gains due to narrow striping, if any, in the presence of inter-stream interference. Note that, narrow striped systems will encounter such interference in practice, since (i) it is difficult to find perfectly anti-correlated streams when collocating stores, or (ii) imprecise workload information at storage allocation time may result in inter-stream interference at run-time.

We then study the impact of load imbalances on the relative performance of wide and narrow striping. Specifically, we consider a narrow striped system where the load on arrays is balanced using the the average load of each stream.

Name	Read req. rate (IOPS)	Write req. rate (IOPS)	Mean request size bytes/request	Num. Request Streams
Web Search 1	334.91	0.07	15509.33	6
Web Search 2	297.48	0.06	15432.40	6
Web Search 3	188.01	0.06	15771.97	6
Financial 1	28.27	93.79	3465.99	24
Financial 2	74.31	15.93	2448.94	19

Table 2: Summary of the Traces. IOPS denotes the number of I/O operations per second.

We then study how dynamic variations in the workload can cause load skews even when the arrays are load balanced based on the mean load. We also study the effectiveness of wide striping in countering such load skews due to its ability to disperse load across all arrays in the system.

Our final experiment is based on a laboratory-based storage system testbed and seeks to validate some of the results obtained from our simulation study. We configure the system for narrow striping and introduce heterogeneous workloads into the system. We then study the behavior of these workloads under wide striping. Our experiment quantifies the performance of the two systems in the presence of interference and load variations.

Together, these three scenarios enable us to quantify the tradeoffs of the two approaches along various dimensions. We now discuss the characteristics of the workloads used in our study.

Workload characteristics: We use a combination of synthetic workloads and real-world trace workloads to generate the request streams in our simulation study. Whereas trace workloads are useful for understanding the behavior of wide and narrow striping in real-world scenarios, synthetic workloads allows us to systematically explore the parameter space and quantify the behavior of the two techniques over a wide range of system parameters. Consequently, we use a combination of these workloads for our study.

Our synthetic workloads are generated using a Poisson ON-OFF process for each request stream. The use of a ON-OFF process allows us to carefully control the amount of interference between streams. Two streams are anti-correlated when they have mutually exclusive ON periods; they are perfectly correlated when their ON periods are synchronized. The degree of correlation can be varied by varying the amount of overlap in the ON periods of streams.

We use two types of ON-OFF processes in our study—those that issue large requests and those that issue small requests. We assume a mean request size of 2MB for large requests and 64KB for small requests; the request sizes are assumed to be exponentially distributed. Request arrivals (during the ON period) are assumed to be Poisson. Successive requests from a stream are assumed to access random locations on the store.

We also use a collection of trace workloads for our study. Specifically, we use five publicly-available block-level I/O traces[2]; the characteristics of these traces are listed in Table 2. Traces labeled Web Search 1 through 3 are I/O traces from a popular web search engine and consists of mostly read requests. Traces labeled Financial 1 and Financial 2 are I/O workloads from OLTP applications of two large financial institutions and have different mixes of read and write requests. Thus, the traces represent different storage environments and, as shown in the Table, have different characteristics.

[2]See http://traces.cs.umass.edu

3.2 Ideal Narrow Striping versus Wide Striping

3.2.1 Comparison using Homogeneous Workloads

We first compare a load-balanced, interference-free narrow striped system with a wide striped system using homogeneous workloads. For the narrow striped system, we assume a single store on each array and assume that each store is accessed by a Poisson ON-OFF request stream. The mean load on each array is identical. Thus, arrays are load balanced and interference-free (since only one stream accesses each array). In the corresponding wide striped system, all stores span all arrays and requests can interfere with one another. We choose request rates that yield an utilization of around 70-80%. We first experiment with large requests that have a mean request size of 2 MB and a stripe unit size of 1 MB. We repeat each experiment with small requests that have a mean size of 64 KB and a stripe unit size of 32KB.

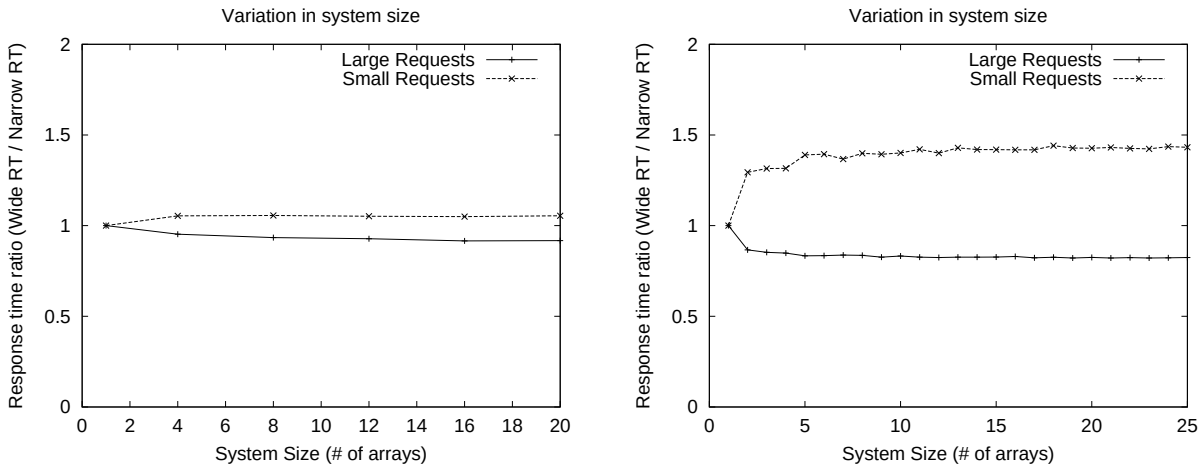
Effect of System Size: Assuming the above setup, we vary the number of arrays in the system from 1 to 25 and measure the response times of requests in the narrow and wide striped system. Each array in the system is accessed by a single stream in narrow striping and all streams access all arrays in wide striping; streams across arrays are assumed to be uncorrelated. Figure 2(a) depicts our results. The figure plots the ratio of the mean response times in the wide and narrow striped systems. A ratio of 1 indicates the two systems are comparable. A ratio smaller than 1 favors wide striping, while that larger than 1 favors narrow striping. Since each array is independent in narrow striping, the variation in the system size does not impact the performance of an individual array in narrow striping. Hence, the variation in the response time ratio is solely due to wide striping.

As shown in Figure 2(a), the performance of the two systems is within 10% of one another over a range of system sizes. Further, wide striping yields better performance for large requests, while narrow striping yields better performance for small requests. The degradation in response times of small requests in wide striping is primarily due to the interference from small requests accessing other stores. For large requests, however, the impact of such interference is outweighed by the increased parallelism available in wide striping, causing wide striping to yield better response time for such requests.

We also varied other parameters such as the stripe unit size, the utilization level, and the request size for homogeneous workloads. In general, we found that the system behavior was similar to that for heterogeneous workloads (discussed in the next section), except that the performance differences become more pronounced in the presence of heterogeneity. Due to space restrictions and to prevent repetition, we only present results for heterogeneous workloads.

3.2.2 Comparison using Heterogeneous Workloads

Having compared narrow and wide striping in the presence of homogeneous workloads, we now compare the two techniques in the presence of heterogeneous workloads. To introduce heterogeneity into the system, we assume that each narrow striped array consists of two stores, one accessed by large requests and the other by small requests (we denote these request streams as L_i and S_i, respectively, where i denotes the i^{th} array in the system). In case of narrow striping, we ensure that only one of these streams is active at any given time. This is achieved by assuming that L_i and S_i are anti-correlated (have mutually exclusive ON periods). Consequently, like before, the narrow striped system is load-balanced and free of inter-stream interference. The wide striped system, however, sees a heterogeneous workload due the simultaneous presence of small and large requests.



(a) Homogeneous workload (b) Heterogeneous workload

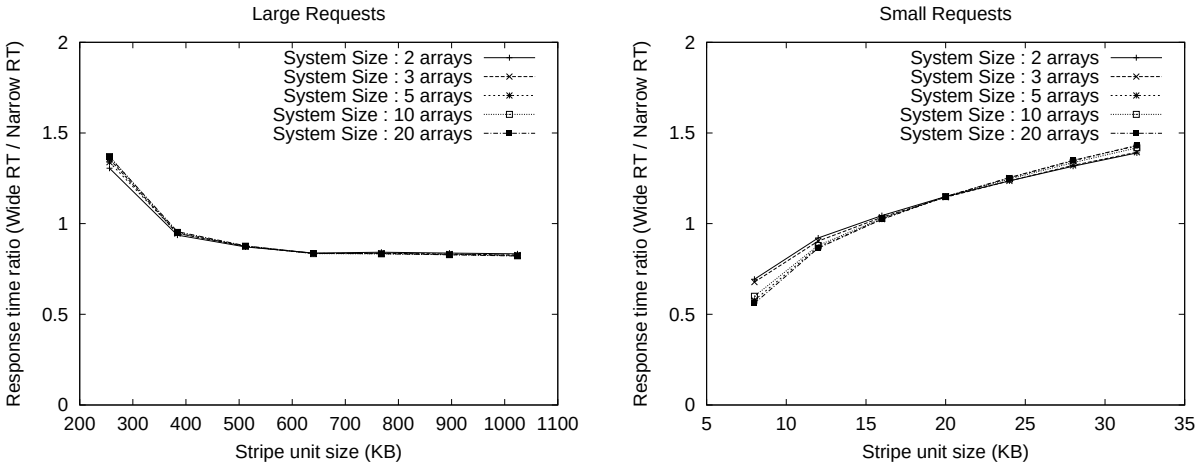
Figure 2: Effect of System Size.

Like before, we assume a mean request size of 2MB for large requests and 64KB for small requests. The default stripe unit size is 1MB and 32 KB for the corresponding stores.

Effect of System Size: We vary the number of arrays in the system from 1 to 25 and measure the average response time of the requests for wide and narrow striping. Figure 2(b) plots the ratio of the average response times for the two systems. The figure shows that while large requests see better response times in wide striping, the opposite is true for small requests. To understand this behavior, we note that two counteracting effects come into play in a wide-striped system. First, there is increased I/O parallelism, since a larger number of disks can simultaneously serve a request and reduce the response time. Second, requests see additional queues that they would not have seen in a narrow striped system, which increases the response time. This is because wide-striped streams access all arrays and interfere with one another. Hence, a small request might see another large request ahead of it, or a large request might see another large request from an independent stream, neither of which can happen in a narrow striped system. Our experiment shows that, for large requests, the benefits of I/O parallelism dominate the degradation due to the interference; this is primarily due to the large size of the requests. For small requests, the interference effect dominates (since a large request can substantially slow down a small request), causing wide striping to yield worse performance for such requests. Figures 2(a) and (b) show that the behavior of narrow and wide striped systems for heterogeneous workloads is identical to that for homogeneous workloads, except that the performance difference is more pronounced due to heterogeneity—depending on the system size, the improvement for large requests ranges from 15-20%, while the degradation for small requests varies from 30-42%.

Effect of Stripe Unit Size: In this experiment, we evaluate the impact of the stripe unit size in wide and narrow striping. We vary the stripe unit size from 128KB to 1MB for large requests, and simultaneously vary the stripe unit size from 8KB to 32KB for small requests. All other parameters are same as the previous experiment. Figures 3(a) and 3(b) plot the response time ratios for large and small requests, respectively. As shown in figures, the stripe unit size has different implications on large and small requests. We consider each scenario separately.

- *Large requests:* Let us consider two extreme stripe units sizes: 128KB and 1MB. Since the mean request size is 2MB, a 128KB stripe unit size causes an average of 16 blocks to be accessed per request. In case of narrow



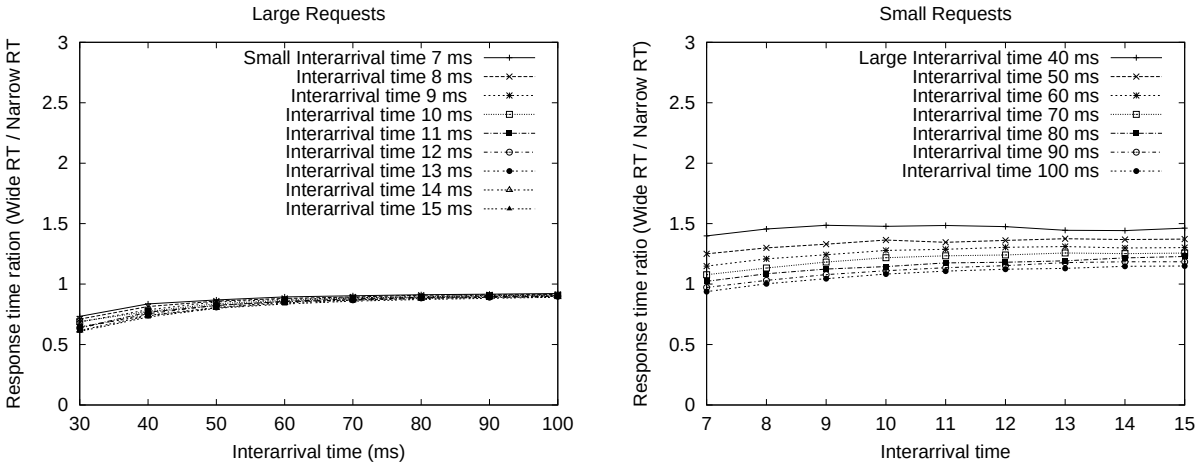
(a) Impact on Large Requests (b) Impact on Small Requests

Figure 3: Effect of the Stripe Unit Size.

striping, since each stream is striped across a 4+p array, multiple blocks are accessed from each disk by a request. Since these blocks are stored contiguously, the disks benefit from sequential accesses to large chunks of data, which reduces disk overheads. In wide striping, each 2MB requests accesses a larger number of disks, which reduces the number of sequential accesses on a disk. Consequently, narrow striping outperforms wide striping for a 128KB unit size. However, as the stripe unit size is increased, the disk overheads become comparable in the two systems (since the benefits of sequential accesses decrease). In particular, with a 1MB stripe unit size, each 2MB request accesses only two disks, and no benefits of sequential accesses accrue in narrow striping. The benefits of I/O parallelism dominate in this case, causing wide striping to outperform narrow striping.

- *Small requests:* Three different factors influence the response times for small requests: disk overheads, I/O parallelism and interference from large requests. At small stripe unit sizes, the disk overhead (seek plus rotational latency) is comparable to the data transfer time for a disk block (these overheads are comparable to the transfer time even when 2-3 blocks are accessed sequentially). Due to these higher disk overheads, a system with higher I/O parallelism yields better response times (but at the expense of higher utilization). Consequently, we see that wide striping outperforms narrow striping for small stripe unit sizes. The two systems incur similar overheads for larger stripe unit sizes. However, as the stripe unit size increases, the interference from large requests also increases considerably (recall that, the stripe unit sizes for both large and small requests is increased in tandem, resulting in larger disk block accesses for L_i streams and increased interference). The interference effect dominates at large stripe unit sizes, yielding worse performance in wide striping.

Effect of the Utilization Level: In this experiment, we study the impact of the utilization level on the response times of wide and narrow striping. We vary the utilization level by varying the inter-arrival times of requests. We first vary the inter-arrival times of large requests from 30ms to 100ms and measure the response times for different background load from small requests. We then vary the inter-arrival times of small requests from 7ms to 15ms and measure the response times for varying background loads from large requests. The various combinations of inter-interval times and background loads result in utilizations ranging from 20% to 60%. Figures 4(a) and 4(b) plot response time ratio for large and small requests, respectively. The figure demonstrates that the wide striped system



(a) Large requests (b) Small Requests

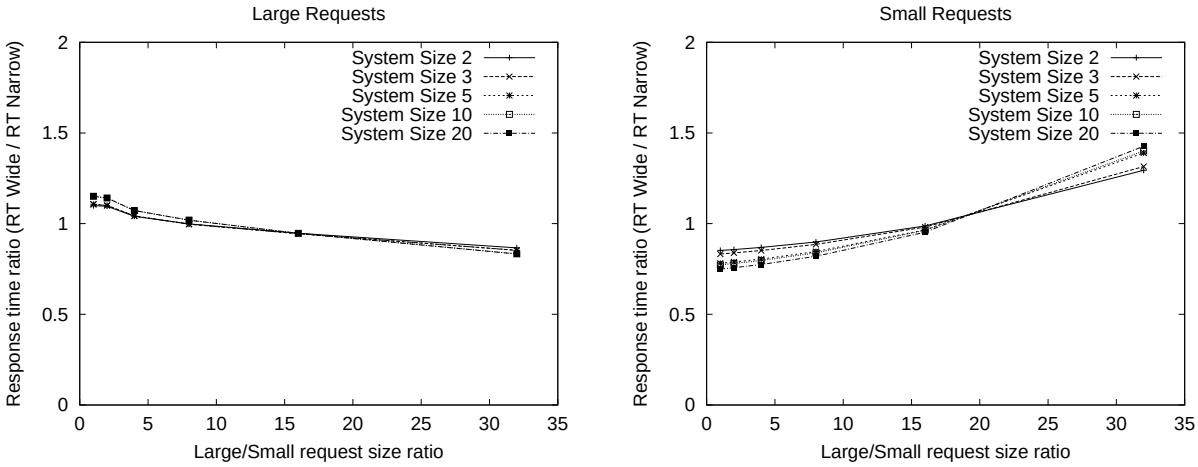
Figure 4: Effect of Utilization Level

performs better with decreasing inter-arrival times (i.e., with increasing utilization). This is because, at high utilization levels, the effects of striping across a larger number of arrays dominate the effects of interference, yielding better response times in wide striping (i.e., the larger number of arrays yield better statistical multiplexing gains and higher I/O parallelism in wide striping). In general, we find that narrow striping outperforms wide striping for small requests sizes and at low utilization levels. In contrast, wide striping yields better performance for large requests; wide striping also yields better performance at high utilization levels regardless of the request size.

Effect of Request Size: In this experiment, we study the impact of the request size on the performance of wide and narrow striping. The experiment was conducted by fixing the inter-arrival times of large and small requests to 40ms and 7ms, respectively. Since large results cause the most interference, we vary the request sizes of large requests and keep the size of small requests fixed. The mean request size for the small requests was chosen to be $64KB$. The average request size for large requests was varied from from $64KB$ to $2MB$. The stripe unit size was chosen to be half the average request size. Figures 5(a) and 5(b) plot response time ratio for large and small requests, respectively, for different system sizes. The X-axis in the figures is the request size ratio for large and small requests (a ratio of 1 is for the case where both request sizes are 64KB, while a ratio of 32 indicates a size of 2MB and 64KB for large and small requests, respectively).

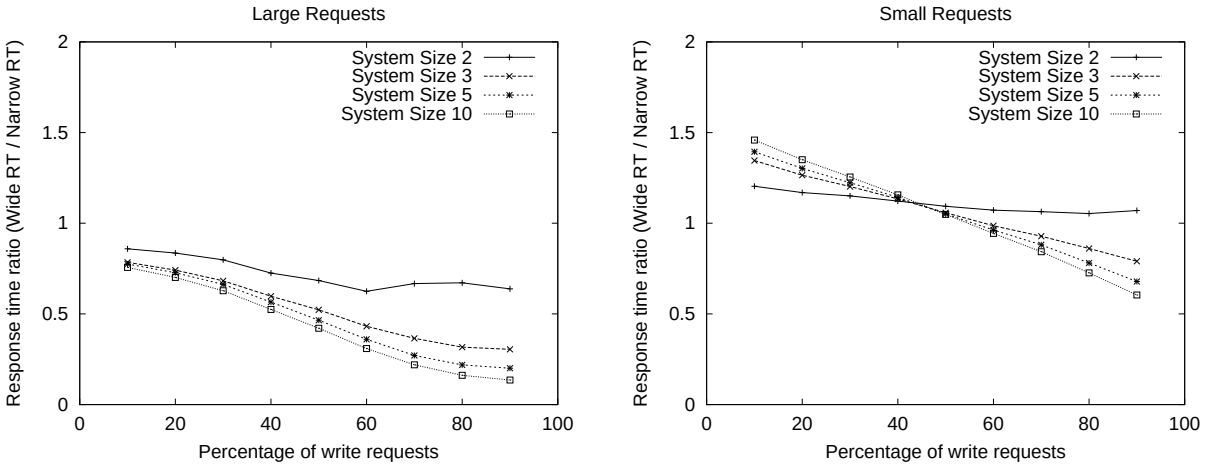
The figures demonstrate that for L_i streams, initially (i.e., at small request sizes), the effect of interference is dominant in wide-striping. However, as the requests become larger, the increased parallelism and load balancing lead to better performance in wide-striping. On the other hand, for small requests, initially the increased parallelism and load-balancing are dominant. However, as the size of the large requests increases, the interference effects start dominating, yielding worse response times in wide striping. At request size ratio of 1, the difference between the L_i and S_i streams is only due to the difference in arrival rates; the L_i streams have a lower arrival rate than the S_i streams.

Effect of Writes: The above experiments have focused solely on read requests. In this experiment, we study the impact of write requests by varying the fraction of write requests in the workload. We vary the fraction of write requests from 10% to 90% and measure their impact on the response times in the wide and narrow striped systems. Figures 6(a) and (b) plot response time ratio for large and small requests, respectively, for different system sizes. Observe that, the presence of write requests increases the effective utilization of the system, since the parity block



(a) Impact on L_i streams (b) Impact on S_i streams

Figure 5: Effect of Request Size



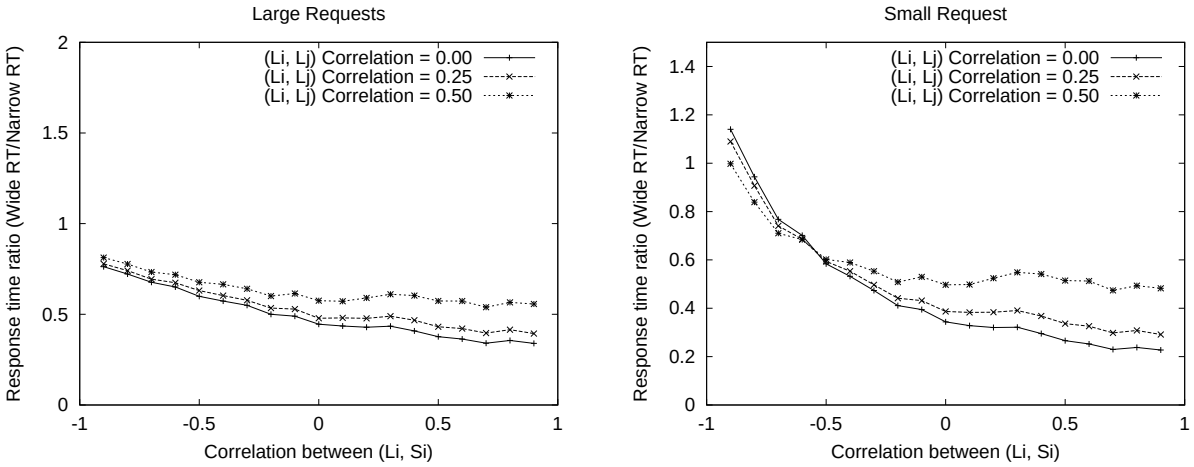
(a) Large requests (b) Small requests

Figure 6: Effect of Write Requests

needs to be updated on a write. For full stripe writes, an additional parity block needs to be written. For small requests, the increase in utilization is even higher, since a read-modify-write cycle is involved.[3] In either case, the increased utilization favors wide striping (for reasons similar to those in the utilization experiment—see Figure 4). In general, the larger the fraction of write requests, the higher is the utilization (as compared to a read-only workload), and the better is the performance of wide striping.

Results Summary: The above experiments study the impact of various system parameters on an ideal narrow striped system and a comparable wide striped system. Our results shows if the stripe unit size is chosen appropriately, then at low utilization levels, streams with large requests perform better in wide-striping. However, this comes at the expense of worse performance for small requests. However, at high utilization levels, the load-balancing effects of

[3]Instead of reading the rest of the parity group, an intelligent array controller can read just the data block(s) being overwritten and the parity block to reconstruct the parity for the remaining data blocks. We assume that the array dynamically chooses between a read-modify write and this reconstruction write strategy depending on which of the two requires fewer reads.



(a) Large requests (b) Small requests

Figure 7: Impact of inter-stream interference.

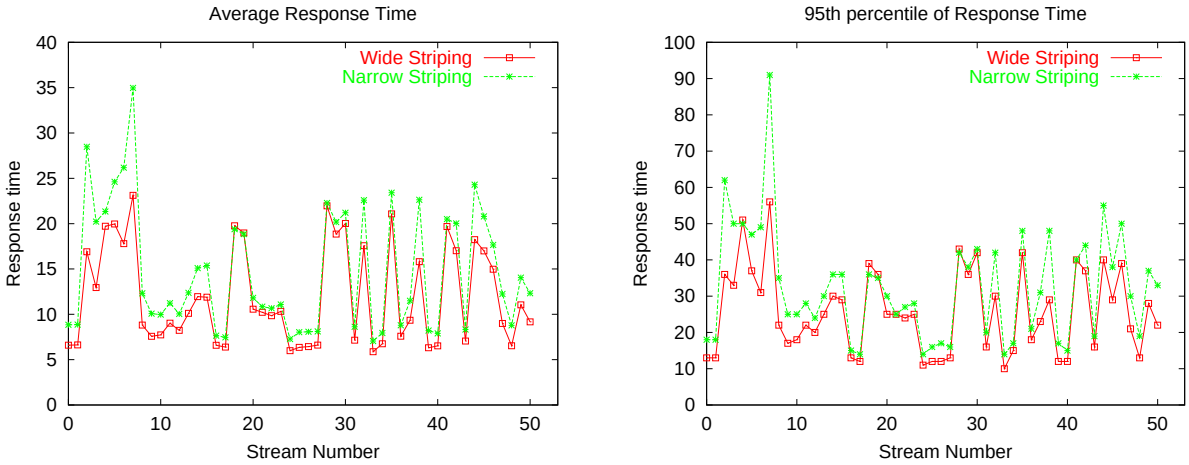
wide striping become dominant and both large and small requests streams perform better in wide striping.

3.3 Impact of Inter-Stream Interference

While our experiments thus far have assumed an ideal (interference-free, load-balanced) narrow-striped system, in practice, storage systems are neither perfectly load balanced nor interference-free. In this section, we examine the impact of one of these dimensions—inter-stream interference—on the performance of narrow (and wide) striping. The next section will study the impact of the other dimension, namely load imbalances, on system performance.

To introduce interference into the system, we assume a narrow striped system with two request streams, L_i and S_i, on each array. Each stream is an ON-OFF Poisson process and we vary the amount of overlap in the ON periods of each (L_i, S_i) pair. Doing so introduces different amounts of correlations (and interference) into the workload accessing each array. Initially, streams accessing different arrays are assumed to be uncorrelated (thus, S_i and S_j as well as L_i and L_j are uncorrelated for all i, j.). Like before, all streams access all arrays in wide striping. We vary the correlation between each (L_i, S_i) pair from -1 to +1 and measure its impact on the response times of large and small requests. Next, we introduce correlations between streams accessing independent arrays in narrow striping (i.e., between (L_i, L_j) and (S_i, S_j) pairs) and again study the effect of correlations between each (L_i, S_i) pair.

Figure 7(a) and (b) plots the resulting response time ratios for large and small requests, respectively. As shown in the figure, the greater the correlation between streams accessing an array, the greater the interference, and the worse is the performance of narrow striping. In general, we find that large requests see worse performance in narrow striping regardless of the correlations. Small requests see better performance only when streams accessing an array are strongly anti-correlated. The response times of small requests worsens substantially as the correlations (and thus the interference) increase. In general, the presence of inter-stream interference benefits wide striping, since its load balancing advantage dominates, yielding better performance. The relative benefits of wide striping decrease as the amount of correlation between streams of the same type increases (e.g., between (L_i, L_j)). Since these streams access independent arrays in narrow striping, the only effect of such correlations is to increase the interference between streams in wide striping, which reduces the relative advantage of wide striping over narrow striping.



(a) Average response time (b) 95^{th} percentile of the response time

Figure 8: Comparison of response time in trace driven simulations.

3.4 Impact of Load Skews: Trace-driven Simulation

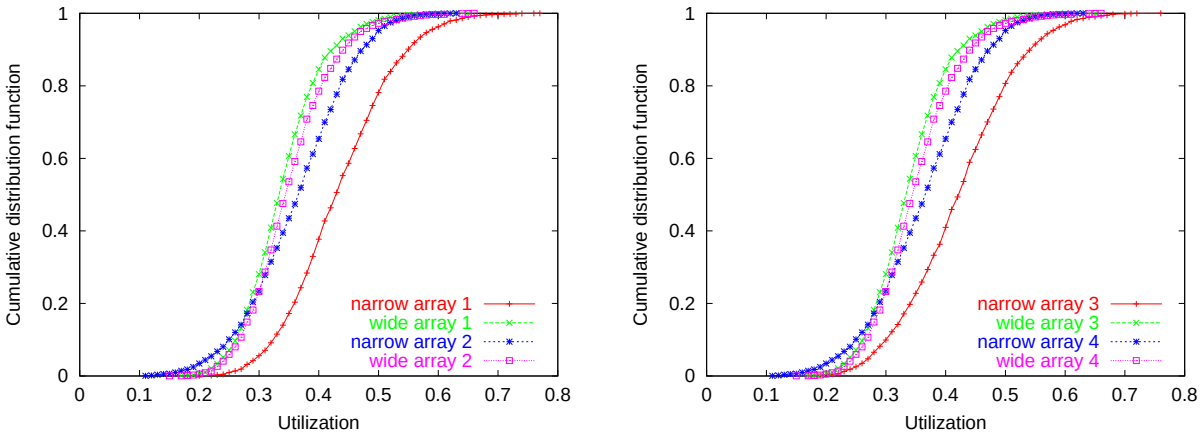
We use the trace workloads listed in Table 2 to evaluate the impact of load imbalance on the performance of narrow and wide striping. The traces have a mix of read and write I/Os and small and large I/Os. To illustrate, the Financial-1 trace has a large fraction of small writes (mean request size is 2.5KB), while the Web-Search-1 trace consists of large reads (mean request size is 15.5KB). Our simulation setup is same as the previous sections, except that each request stream is driven by our traces instead of a synthetic ON-OFF process.

To compare the performance of narrow and wide-striping using these traces, we separate each independent stream from the trace (each stream consists of all requests accessing a volume). This pre-processing step yields 61 streams. We then eliminate 9 streams from the search engine traces, since these collectively contained less that 1000 requests (and are effectively inactive). We then partition the remaining 52 streams into four sets such that each set is load-balanced. We use the write-weighted average IOPS[4] of each stream as our load balancing metric—in this metric, each write request is counted as four I/Os (since each write could, in the worst case, trigger a read-modify-write operation involving four I/O operations). Since the size of each I/O operation is relatively small, we did not consider stream bandwidth as a criteria for load balancing.

We employ a greedy algorithm for partitioning the streams. The algorithm creates a random permutation of the streams and assigns them to partitions one at a time, so that each stream is mapped to the partition that results in the least imbalance (the imbalance is defined to the difference between the load of the most heavily-loaded and the least lightly-loaded partitions). We repeat the process (by starting with another random permutation) until we find a partitioning that yields an imbalance of less than 1%.

Assuming a system with four RAID-5 arrays, each configured with $4 + p$ disks, we map each partition to an array in narrow striping. All partitions are striped across all four arrays in wide striping. We computed the average response time as well as the 95^{th} percentile of the response times for each stream in two systems. Figure 8(a) and (b) plot the average response time and the 95^{th} percentile of the response time, respectively, for the various streams (the X axis is the stream id). As shown in the figure, wide striping yields average response times that are comparable to or better

[4]I/O Operations Per Second



(a) Arrays 1 and 2 (b) Arrays 3 and 4

Figure 9: The cumulative distribution function (CDF) of the array utilizations in wide and narrow striping.

than a narrow striped system for most streams. Furthermore, the improvement is larger for the 95^{th} percentile of the response time. This is primarily due to the better load balancing ability of wide striping—each stream introduces a time-varying workload and wide striping is able to disperse load increases to a larger number of disks, yielding better response times during load skews. The load balancing properties of wide striping are illustrated in Figures 9(a) and 9(b), which plot the cumulative distribution function (CDF) of the utilizations of the four arrays in the narrow and wide striping systems. As shown in the figures, all the four arrays are equally balanced in wide striping. However, the utilization distributions are significantly different for all the four arrays in the narrow striping case. This demonstrates that, even though the arrays are balanced using the average load, significant variations in the load occur over time, causing the per-array utilizations to differ from one another.

While the above experiment shows that wide striping outperforms narrow striping at high utilization levels and in the presence of load skews, we conduct an additional experiment to compare the two techniques at low utilizations. We begin with the same setup as the previous experiment. After approximately one hour of simulation time, we terminate two high-intensity streams accessing array 2 in narrow striping. Hence, this array sees a very low utilization ($< 10\%$) for the remaining 9 hours of the simulation time; the other three arrays continue to operate at moderate to high utilization levels. We then repeat this experiment for wide striping. Figure 10 plots the response time ratios for each array. As shown in the figure, due to the very low load on array 2, streams accessing this array see better response times in narrow striping. In wide striping, since all streams access all arrays, all streams benefit from the reduced utilization, but the magnitude of the response time reduction for any given stream is small. Like in the previous experiment, wide striping continues to yield comparable or better response times for the remaining streams. These results are consistent with those obtained from synthetic workloads—narrow striping outperforms wide striping at low utilization levels and for small requests, while wide striping yields better performance at high utilization levels or in the presence of load skews.

3.5 Experiments with a Disk Array

Our final experiment involves a comparison of narrow and wide striping on a laboratory storage-system testbed. Our testbed consists of a four-processor IBM RS6000 machine with 512 MB RAM and AIX 4.3.3. The machine contains

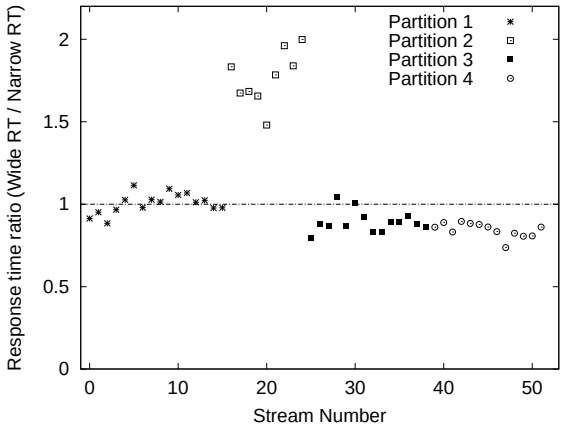


Figure 10: Response times ratios for the streams when array 2 is under-utilized.

a SSA RAID adapter card with two channels (also called SSA loops) and 16 9GB disks on each channel (total of 32 disks). We configured four RAID-5 arrays, two arrays per channel, each in a $7+p$ configuration. Whereas two of these arrays are used for our narrow striping experiment, the other two are used for wide striping (thus, each experiment uses 16 disks in the system). The SSA RAID card uses a stripe unit size of 64 KB on all arrays; the value is chosen by the array controller and can not be changed by the system. However, as explained below, we use large requests to emulate the behavior of larger stripe unit sizes in the system. We use two workloads in our experiments:

- *TPC-C benchmark*: The TPC-C benchmark is an On-Line Transaction Processing (OLTP) benchmark and results in mostly small size random I/Os. The benchmark consists of a mix of reads and writes (approximately two-thirds reads and one-third writes [3]). We use a TPC-C setup with 300 warehouses and 30 clients.

- *Large sequential*: This is an application that reads a raw volume sequentially using large requests. The process has an I/O loop that issues requests using the $read()$ system call. Since we can not control the array stripe unit size, we emulate the effect of large stripe units by issuing large read requests. We use two request sizes in our experiments. To emulate a 128KB stripe unit size, we issue 896KB requests (since $64KB \times 7disks = 448KB$, a 898 KB request will access two 64 KB chunks on each disk). We also find experimentally that the throughput of the array is maximized when requests of $448KB \times 16 = 7$ MB are issued in a single $read$ call. Hence, we use $7MB$ as the second request size (which effectively requests 16 64KB blocks from each disk).

We first experiment with a narrow striped system by running the TPC-C benchmark on one array and the sequential application on the other array. We find the TPC-C throughput to be 267 TpmC, while the throughput of the sequential application is 25.43 MB/s for 896 KB requests and 29.45 MB/s for 7MB requests (see Table 3).

We then experiment with a wide striped system. To do so, we create three logical volumes on the two arrays using the AIX volume manager. Two of these volumes are used for the TPC-C data, index, and temp space, while the third volume is used for the sequential workload. As shown in Table 3, the TPC-C throughput is 356 TpmC when the sequential workload uses 896 KB requests and is 218 TpmC for 7MB requests. The corresponding sequential workload throughput is 20.09 MB/s and 36.86 Mb/s, respectively.

Striping	Sequential I/O Size	Sequential Throughput	TPC-C Throughput
Narrow	896 KB	25.43 MB/s	267 TpmC
Wide	896 KB	20.09 MB/s	356 TpmC
Narrow	7 MB	29.45 MB/s	267 TpmC
Wide	7 MB	36.86 MB/s	218 TpmC

Table 3: TPC-C and Sequential Workload Throughput in Narrow and Wide Striping

Thus, we find that for the sequential workload, small requests favor narrow striping, while large requests favor wide striping. For TPC-C workload, the reverse is true, i.e., small requests favor wide striping and large requests favor narrow striping. This is because the performance of TPC-C is governed by the interference from the sequential workload. The interference is greater when the sequential application issues large 7MB requests, resulting in lower throughput for TPC-C. There is less interference when the sequential application issues 898 KB (small) requests; further, TPC-C benefits from the larger number of arrays in the wide striped system, resulting in a higher throughput. This behavior is consistent with the experiments presented in previous sections. Furthermore, the performance difference (i.e., improvement/degradation) between the two systems is around 20%, which is again consistent with the results presented earlier.

3.6 Summary and Implications of our Experimental Results

Our experiments show that narrow striping yields better performance when: (1) streams can be ideally partitioned such that the partitions are load-balanced and there is very little interference between streams within a partition; (2) utilization levels are low; and (3) request sizes are small. One or more of these conditions may not hold in real systems. For example, in our trace-driven experiments, we found that even though the average load was balanced, wide-striping performed comparably or better than narrow-striping. With a TPC-C workload, we found that if the stripe unit is chosen appropriately, then narrow and wide striping have comparable performance even though there are no workload skews due to the "constant-on" nature of the benchmark.

In situations where it is beneficial to do narrow striping, significant efforts are required to extract those benefits. First, the workload has to be determined either by specification in the policy or by system measurement. Since narrow placement derives benefits from exploiting the correlation structure between streams, the characteristics of the streams as well as the correlations between the streams needs to be determined. It is not known whether stream characteristics or the inter-stream correlations are stable over time. Hence, if the assumptions made by the narrow placement technique change, then load imbalances and hot-spots may occur. These hot-spots have to be detected and the system re-optimized using techniques such as [4]. This entails moving stores between arrays to achieve a new layout [18]. The process of data movement itself has overheads that can effect the performance. Furthermore, data migration techniques are only useful for long-term or persistent workload changes; short-time scale hot-spots that occur in modern systems can not be effectively resolved by such techniques. Thus, it is not apparent it is possible to extract the benefits of narrow-striping in real systems. Policy-managed systems that employ narrow striping [3, 4] have only compared performance with manually-tuned narrow striped systems. While these studies have shown that such systems can perform comparably or outperform human-managed narrow striped systems, no comprehensive comparison with wide striping was undertaken in these efforts.

In contrast to narrow striping, which requires detailed workload knowledge, the only critical parameter in wide striping seems to be the stripe unit size. Our experiments highlight the importance of choosing an appropriate stripe unit for each store in a wide striping system (for example, large stripe units for streams with large requests). While an optimal stripe unit size may itself depend on several workload parameters, our preliminary experiments indicate that choosing the stripe unit size based on the average request size is a good rule of thumb. For example, in our experiments, we chose the stripe unit to be half the average request size. Detailed analytical and empirical models for determining the optimal stripe unit size also exist in the literature [8, 9, 21].

A policy-based storage management system must also consider issues unrelated to performance when choosing an appropriate object placement technique. For example, system growth has different consequences for narrow and wide striping. In case of narrow striping, when additional storage is added, data does not have to be necessarily moved; data needs to move only to ensure optimal placement. In case of wide-striping, data on all stores needs to be reorganized to accommodate the new storage. Although this functionality can be automated and implemented in the file system, volume manager, or raid controllers without requiring application down-time, the impact of this issue depends on the frequency of system growth. In enterprises environments, system growth is usually governed by purchasing cycles that are long. Hence, we expect this to be an infrequent event and not be a significant issue for wide-striping. In environments where system growth is frequent, however, such data reorganizations can impose a large overhead.

A policy based storage management system may also be required to provide different response time or throughput guarantees to different applications. The choice between narrow and wide striping in such a case would depend on the Quality of Service (QoS) control mechanisms that are available in the storage system. For example, if appropriate disk scheduling mechanisms exist in the storage system [22], then it may be desirable to do wide striping. If no QoS control mechanisms exist, a system can either isolate stores using narrow striping, or group stores with similar QoS requirements, partition the system based on storage requirements of each group, and wide-stripe each group within the partition.

4 Related Work

The design of policy-managed storage systems was pioneered by [3, 5, 4, 18], where techniques for automatically determining storage system configuration were studied. This work determines: (1) the number and types of storage systems that are necessary to support a given workload, (2) the RAID levels for the various objects, and (3) the placement of the objects on the various arrays. The placement technique is based on narrow striping. It exploits the accesses correlation between various objects and stores bandwidth-bound and space-bound objects together to determine an efficient placement. The focus of our work is different; we assume that the number of storage arrays as well as the RAID levels are predetermined and study the suitability of wide and narrow striping for policy-managed systems.

Analytical and empirical techniques for determining file-specific stripe unit, placing files on disk arrays, and cooling hot-spots have been studied in [8, 9, 16, 21]. Our work addresses a related but largely orthogonal question of the benefits of wide and narrow striping for policy-managed storage systems.

While much of the research literature has implicitly assumed narrow striping, practitioners—database and storage administrators—are increasingly advocating wide striping due to its inherent simplicity. For instance, wide-striping combined with mirroring has been advocated for database workloads in [17]. A cursory evaluation of this technique,

referred to as *Stripe and Mirror Everything Everywhere (SAME)*, has been presented in [1]; the work uses a simple storage system configuration to demonstrate that wide striping can perform comparably to narrow striping. To the best of our knowledge, ours is the first work that systematically evaluates the tradeoffs of wide and narrow striping.

The performance benefits of interleaving video data and conventional data on the same set of disks, instead of separating them onto different disks, have been evaluated in [22]. Like our work, they examine the effectiveness of narrow and wide striping, albeit for heterogeneous data types. However, the focus of [22] is primarily on QoS-related issues. Specifically, disk scheduling algorithms that provide QoS guarantees to soft real-time video requests without significantly affecting the performance of textual requests are studied for wide striped systems. In contrast, we study the impact of load skews and inter-stream interference for wide and narrow striped systems using conventional workloads.

Several orthogonal issues in the design of large storage systems have also been studied in the literature. Techniques for characterizing I/O workloads have been studied in [12, 14]. Techniques for making storage devices more intelligent have been studied in [7, 13]. Techniques for load-balancing and handling workload skews in disk arrays using chained declustering have been studied in [11]. Randomized data placement techniques have been compared to traditional striping techniques in [20]. We note that our results for wide striping and narrow striping hold even for environments that use randomized data placement instead of striping—our experiments (not reported here) show similar tradeoffs when objects are randomly placed on a shared set of arrays instead of being partitioned onto separate arrays.

5 Concluding Remarks

Storage management cost is a significant fraction of the total cost of ownership of large, enterprise storage systems. Consequently, software automation of common storage management tasks so as to reduce the total cost of ownership is an active area of research. In this paper, we considered a policy-managed storage system—a system that automates various management tasks—and focused on the problem of the storage allocation techniques that should be employed in such a policy-managed system. We studied two fundamentally different storage allocation techniques for policy-managed systems: narrow and wide striping. Whereas wide striping techniques need very little workload information for making placement decisions, narrow striping techniques employ detailed information about the workload to optimize the placement and achieve better performance. We systematically evaluated this trade-off between "information" and performance. Using synthetic and real I/O workloads, we found that an idealized narrow striped system can outperform a comparable wide-striped system at low levels of utilization and for small requests. However, wide striping outperforms narrow striped system when the system utilization is high or in the presence of workload skews that occur in real systems. Our experiments demonstrate that the additional workload information needed by narrow placement techniques may not necessarily translate to better performance. Based on our results, we advocate narrow striping only when (i) the workload can be characterized precisely a priori, (ii) the system can be sufficiently provisioned to keep the utilization low, and (iii) it is feasible to use data migration to handle workload skews and workload interference. In general, we recommend that (i) policy-managed systems use wide striping for object placement and (ii) sufficient information be specified at storage allocation time to enable appropriate selection of the stripe unit size.

References

[1] Configuring the Oracle Database with Veritas Software and EMC Storage. Technical report, Oracle Corporation. Available from http://otn.oracle.com/deploy/availability/pdf/ora_cbook1.pdf.

[2] N. Allen. Don't Waste Your Storage Dollars. Research Report, Gartner Group, March 2001.

[3] G. Alvarez, E. Borowsky, S. Go, T. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: An Automated Resource Provisioning Tool for Large-scale Storage Systems. *ACM Transactions on Computer Systems (to appear)*, 2002.

[4] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: Running Circles Around Storage Administration. In *Proceedings of the Usenix Conference on File and Storage Technology (FAST'02), Monterey, CA*, pages 175–188, January 2002.

[5] E. Anderson, M. Kallahalla, S. Spence, R. Swaminathan, and Q. Wang. Ergastulum: An Approach to Solving the Workload and Device Configuration Problem. Technical Report HPL-SSP-2001-05, HP Laboratories SSP, May 2001.

[6] E. Anderson, R. Swaminathan, A. Veitch, G. Alvarez, and J. Wilkes. Selecting RAID Levels for Disk Arrays. In *Proceedings of the Conference on File and Storage Technology (FAST'02), Monterey, CA*, pages 189–201, January 2002.

[7] A. Brown, D. Oppenheimer, K. Keeton, R. Thomas, J. Kubiatowicz, and D.A. Patterson. ISTORE: Introspective Storage for Data-Intensive Network Services. In *Proceedings of the 7th Workshop on Hot Topics in Operating Systems (HotOS-VII), Rio Rico, AZ*, March 1999.

[8] P. Chen and D. Patterson. Maximizing Performance in a Striped Disk Array. In *Proceedings of ACM SIGARCH Conference on Computer Architecture, Seattle, WA*, pages 322–331, May 1990.

[9] P. M. Chen and E. K. Lee. Striping in a RAID Level 5 Disk Array. In *Proceedings of the 1995 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1995.

[10] R. Flynn and W. H. Tetzlaff. Disk Striping and Block Replication Algorithms for Video File Servers. In *Proceedings of IEEE International Conference on Multimedia Computing Systems (ICMCS)*, pages 590–597, 1996.

[11] L. Golubchik, J. Lui, and R. Muntz. Chained Declustering: Load Balancing and Robustness to Skew and Failures. In *Proceedings of the Second International Workshop on Research Issues in Data Engineering (RIDE), Tempe, Arizona*, pages 88–95, February 1992.

[12] K. Keeton, G. Alvarez, E. Riedel, and M. Uysal. Characterizing I/O-intensive Workload Sequentiality on Modern Disk Arrays. In *Proceedings of the 4th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-01)*, January 2001.

[13] K. Keeton, D A. Patterson, and J. Hellerstein. The Case for Intelligent Disks (IDisks). In *Proceedings of the 24th Conference on Very LArge Databases (VLDB)*, August 1998.

[14] K. Keeton, A. Veitch, D. Obal, and J. Wilkes. I/O Characterization of Commercial Workloads. In *Proceedings of the 3rd Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-00)*, January 2000.

[15] E. Lamb. Hardware Spending Matters. *Red Herring*, pages 32–22, June 2001.

[16] E.K. Lee and R.H. Katz. An Analytic Performance Model for Disk Arrays. In *Proceedings of the 1993 ACM SIGMETRICS*, pages 98–109, May 1993.

[17] J. Loaiza. Optimal Storage Configuration Made Easy. Technical report, Oracle Corporation. Available from http://otn.oracle.com/deploy/performance/pdf/opt_storage_conf.pdf.

[18] C. Lu, G. Alvarez, and J. Wilkes. Aqueduct: Online Data Migration with Performance Guarantees. In *Proceedings of the Usenix Conference on File and Storage Technology (FAST'02), Monterey, CA*, pages 219–230, January 2002.

[19] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Array of Inexpensive Disks (RAID). In *Proceedings of ACM SIGMOD'88*, pages 109–116, June 1988.

[20] J. Santos, R. Muntz, and B. Ribeiro-Neto. Comparing Random Data Allocation and Data Striping in Multimedia Servers. In *Proceedings of ACM SIGMETRICS 2000, Santa Clara, CA*, pages 44–55, June 2000.

[21] P. Scheuermann, G. Weikum, and P. Zabback. Data Partitioning and Load Balancing in Parallel Disk Systems. *VLDB Journal*, 7(1):48–66, 1998.

[22] P. Shenoy, P. Goyal, and H M. Vin. Architectural Considerations for Next Generation File Systems. In *Proceedings of the Seventh ACM Multimedia Conference, Orlando, FL*, November 1999.

[23] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID Hierarchical Storage System. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles, Copper Mountain Resort, Colorado*, pages 96–108, Decmember 1995.