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

Vijay Sundaram, Pawan Goyal†, **Peter Radkov** and **Prashant Shenoy**

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

†Storage Systems Department
IBM Almaden Research Center
goyalp@us.ibm.com

Abstract

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 by such a system. We study two fundamentally different storage allocation techniques: narrow and wide striping. Whereas wide striping stripes each object across all the disks in the system and needs very little workload information for making placement decisions, narrow striping techniques stripe an object across a subset of the disks and employs detailed workload information to optimize the placement. 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 for small requests. However, wide striping outperforms narrow striped systems in the presence of workload skews that occur in real I/O workloads; the two systems perform comparably for a variety of other real-world scenarios. Our experiments show that the additional workload information needed by narrow placement techniques may not necessarily translate to significantly 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 complex systems consisting of tens or hundreds of RAID arrays. The sheer size of these systems, coupled with the diversity of the applications which access them, makes their administration a complex task. The complexity of the administration tasks in turn significantly increases the cost of ownership of the entire system; not surprisingly, management costs are now a significant fraction (75-90%) of the total cost of ownership of such systems [3, 11]. Since many administration tasks are amenable to software automation, we propose to address these problems by designing a policy-managed system; a *policy-managed storage system* is one that automatically allocates storage based on a policy and manages it so that the specified performance objectives are met.

In a policy managed storage 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, requests for allocating storage may specify desired performance constraints on the throughput and the response times. Automation of these and other tasks can significantly reduce the cost and complexity of storage management. 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 [5, 8, 9, 10, 16]. 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 [5, 8, 16]. 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. An alternate approach is to assume that detailed knowledge of the workload is difficult to obtain a priori and to use 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.

Although narrow striping is advocated both by the research literature and widely used in practice, at least one major database vendor has recently advocated the use of wide striping to simplify storage administration [1, 13]. However, no systematic study of the two techniques exists in the literature.

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 striped policy-managed systems are easier to design and use, while narrow-striped policy-managed systems can potentially make better storage decisions. This results in a simplicity versus performance tradeoff— wide striped systems advocate simplicity by requiring less workload information, which can potentially result in worse performance. The opposite is true for a narrow striped system. Narrow striping can extract performance gains only if the workload specification is precise. It is not *a priori* evident if narrow striping can make better storage decisions when the workload specification is imprecise or incorrect (the accuracy of the workload information is not an issue in wide striping, since no such information is required for placement decisions). Although placement of objects in large storage systems have been extensively studied [4, 5, 6, 16], surprisingly, no systematic study of these tradeoffs of wide and narrow striping exists in the literature. Our work seeks to address this issue by answering the following questions:

- 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?
- From a performance standpoint, how do narrow and wide striping compare against one another? What is the impact of interference between requests accessing the same set of disks in wide striping? Similarly, what is the impact of imprecise workload knowledge and the resulting load skews in narrow striping?

We answer these questions using (i) simulations driven by OLTP traces and synthetic workloads and (ii) experiments on a storage system testbed. We find that an idealized narrow striped system can outperform a comparable wide-striped system for small requests. However, wide striping outperforms

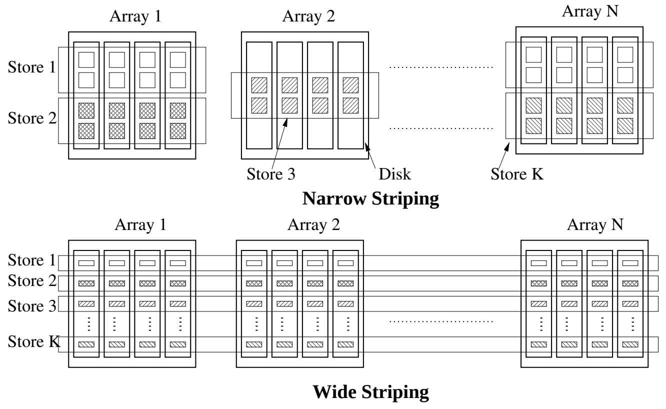


Figure 1: Narrow and wide striping in an enterprise storage system.

narrow striped systems 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 a significant performance advantage. Consequently, we advocate the use of narrow striping only when (i) the workload can be characterized precisely a priori, and (ii) it is feasible to use data migration to handle workload skews and workload interference. In general, we argue for simplicity and 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 disk arrays. A disk array is essentially a collection of physical disks that presents an abstraction of one or more logical disks to the rest of the system. 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 [15]. 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 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 decision support system application may request 15GB of storage space optimized for I/O throughput; similarly, an OLTP application 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* [4]; the data on a store is collectively referred to as a *data object* (e.g., a tablespace, 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 pictorially depicts narrow and wide striping). Since the RAID-level selection problem has been studied in the literature [7, 18], in this paper, we will focus only on the problem of mapping stores onto arrays.

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 interference in narrow striping by mapping stores with anti-correlated access patterns on to a single array. The effectiveness of such optimizations depends 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 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 arrays 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 tradeoffs and to 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 simulations and experiments on a storage system testbed.

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 an 18 GB

[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

Fujitsu MAJ3182MC disk; the characteristics of this disk are shown in Table 1. The disk head movement is modeled as in [9]. We also incorporate a write-back LRU cache to capture the effect of the storage controller cache. The cache size is varied linearly with the number of arrays in the storage system, with 64 MB of cache per array. The cache also employs an early destage policy to evict dirty buffers.

Our storage system testbed consists of a IBM TotalStorage FAStT-700 storage subsystem equipped with 40 18 GB disks. The storage subsystem is connected to a 1.6 GHz Pentium 4 server with 512 MB RAM running Linux 2.4.18 over Fibre Channel. The specific RAID configurations used in our experiments are described in the corresponding experimental sections.

Depending on whether narrow or wide striping is used, each object (and the corresponding store) is either placed on a single array or striped across all arrays in the system. We assume each store is allocated a contiguous amount of space on each 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 an OLTP application is the aggregation of I/O requests triggered by various transactions. We use a combination of synthetic and trace-driven workloads to generate request streams in our simulations; the characteristics of these workloads are 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 zero 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 studying each dimension—we first vary *the interference between request streams and then the load imbalance.*

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. 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 set of experiments compare the performance of narrow and wide striping using two well-known database benchmarks—TPC-C and TPC-H. We also study the effects of interference and load variations on the two systems.

Together, these 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, real-world traces and database benchmarks to generate the request streams in our study. Whereas trace workloads are useful for understanding the behavior of wide and narrow striping in real-world scenarios, synthetic workloads allow us to systematically explore the parameter space and quantify the behavior of the two techniques over a wide range of system parameters. Database benchmarks, on the other hand, allow for comparisons based on "standardized" workloads. Consequently, we use a combination of these workloads for our study.

Our synthetic workloads are generated using two types of processes: (1) *Poisson ON-OFF process:* The on and off periods of such a process are exponentially distributed. Request arrivals during the ON period are assumed to be Poisson. Successive requests are assumed to access random locations on the store. The use of an 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.

Name	Read req. rate (IOPS)	Write req. rate (IOPS)	Mean req. size (bytes/req)	Request Streams
OLTP 1	28.27	93.79	3466	24
OLTP 2	74.31	15.93	2449	19
Web Search 1	334.91	0.07	15509	6
Web Search 2	297.48	0.06	15432	6
Web Search 3	188.01	0.06	15772	6

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

(2) *Closed-loop process:* A closed-loop process with *concurrency* N consists of N concurrent clients that issue requests continuously, i.e., each client issues a new request as soon as the previous request completes. The request sizes are assumed to be exponentially distributed and successive requests access random locations on the store.

Both the Poisson ON-OFF and closed-loop processes can generate two types of request streams—those that issue small requests and those that issue large requests. Streams with large requests are representative of decision support systems (DSS), while those with small requests represent OLTP applications. Since DSS workloads access large amounts of data, we assume a mean request size of 1MB for large requests. On the other hand, since OLTP applications generate small requests, we use 4KB for small requests; the request sizes are assumed to be exponentially distributed. Prior studies have used similar parameters [4]. The stripe unit sizes of the stores being accessed by large and small requests was set to be 512KB and 4KB, respectively.

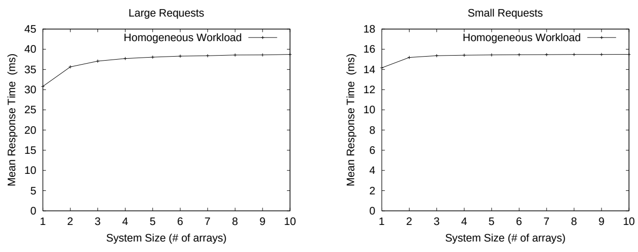
We also use a collection of block-level I/O trace workloads for our study; the characteristics of these traces are listed in Table 2. Traces labeled OLTP 1 and OLTP 2 are I/O workloads from OLTP applications of two large financial institutions and have different mixes of read and write requests. Traces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

3.2 Ideal Narrow Striping versus Wide Striping

We first compare a load-balanced, interference-free narrow striped system with a wide striped system using homogeneous and heterogeneous workloads. In case of homogeneous workload, all streams generate requests of similar sizes. In case of heterogeneous workload, streams generate requests of different sizes.

3.2.1 Comparison using Homogeneous Workloads

We compare narrow and wide striping, first for small request sizes and then for large requests. Our simulations assume that each narrow striped array consists of a single store (and a single request stream), while all stores are striped across all arrays in wide striping. We use closed-loop workloads to generate requests streams; the concurrency factor for each large and small closed-loop workload was assumed to be 2 and 4,



(a) Large Requests (b) Small Requests

Figure 2: Effect of system size for homogeneous closed-loop workloads. System size of 1 depicts narrow striping.

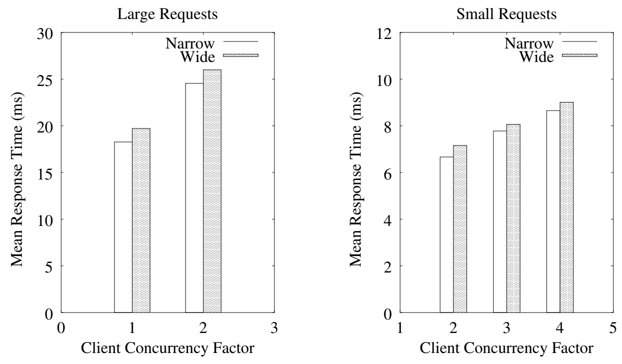
respectively. We vary the number of arrays in the system, i.e., the system size, and measure the average response time in the two systems. Figures 2(a) and 2(b) depict the response times for large and small request sizes, respectively, in the two systems. When the system size is 1 (i.e., a single array accessed by a single stream), narrow and wide striping are identical. Further, since each request stream accesses a different array in narrow striping, the system size has no impact on the response time. In other words, the performance of narrow striping is represented by the system size of 1 (and remains unchanged). In contrast, the response time of wide striping degrades with increasing system sizes. This is primarily due to increased interference between request streams. However, as shown in Figure 2, the impact of such interference increases slowly with system size. Overall, we find that wide striping sees response times that are 10-20% worse than narrow striping.

We validate the results of the above simulation experiment using our FAStT-700 storage testbed. We configure the FAStT with two RAID-5 arrays (4+p configuration). We create two stores, each 2 GB in size, on the storage system. For large requests, the stripe unit size of the store is 256 KB and for small requests it is configured to be 8 KB. We used the *Linux Logical Volume Manager* (LVM) for wide striping the stores. The mean request size for large and small requests is chosen to be 512 KB and 8 KB, respectively. We compare narrow and wide striping using closed-loop workloads with different concurrency factors (see Figure 3). As Figure 3 demonstrates, the response time in the wide-striped system is about $10-15\%$ higher than in the narrow-striped system which is consistent with the results of our simulations.

In addition to the above experiments, we compared narrow and wide striping by varying a variety of system parameters such as the stripe unit size, the request size, the utilization level, and the percentage of write requests. Our experiments were carried out for both closed-loop and the open-loop Poisson OF-OFF workloads. In each case, we found that, if the stripe unit size is chosen carefully, the performance of narrow and wide striped systems is comparable and within $10-15\%$ of one another. Due to lack of space, we omit the results from these experiments, which can be found in Appendix A.

3.2.2 Comparison using Heterogeneous Workloads

To introduce heterogeneity into the system, we assume that each narrow striped array consists of two stores, one accessed



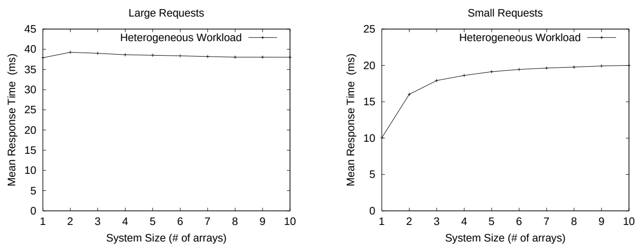
(a) Large Requests (b) Small Requests

Figure 3: Homogeneous Workload: Closed-loop Testbed Experiments

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). We do not assume any correlations between streams accessing independent arrays (i.e, between streams L_i and L_j or S_i and S_j). 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.

We use Poisson On-Off processes to understand the effect of various parameters such as system size, stripe unit size, utilization level, etc. As before, we assume a mean request size of 1MB for large requests and 4KB for small requests. The default stripe unit size is chosen to be 512KB and 4KB for the corresponding stores. Unless specified otherwise, we chose request rates that yield utilization of around 60-65%; this corresponds to a mean inter-arrival (*IA*) time of 17 ms for large requests and 4 ms for small requests, respectively.

Effect of System Size: We vary the number of arrays in the system from 1 to 10 and measure the average response time of the requests for wide and narrow striping. Since each array is independent in narrow striping, the system size has no impact on the performance of an individual array. Hence, like before, the performance of narrow striping is represented by a system size of 1 (and remains unchanged, regardless of the system size). Figures 4(a) and 4(b) show the response times on large and small requests, respectively, for varying system sizes. The figure shows that while large requests see comparable response times in wide striping, small requests see worse performance. To understand this behavior, we note that two counteracting effects come into play in a wide-striped system. First, since stores span all arrays, there is better load balancing across arrays, yielding smaller 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, as one increases the system size, the benefits of better load-balancing balance the slight 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), leading to a higher response time in wide striping. Observe that response time is higher by approximately the transfer time of a stripe unit of large request.

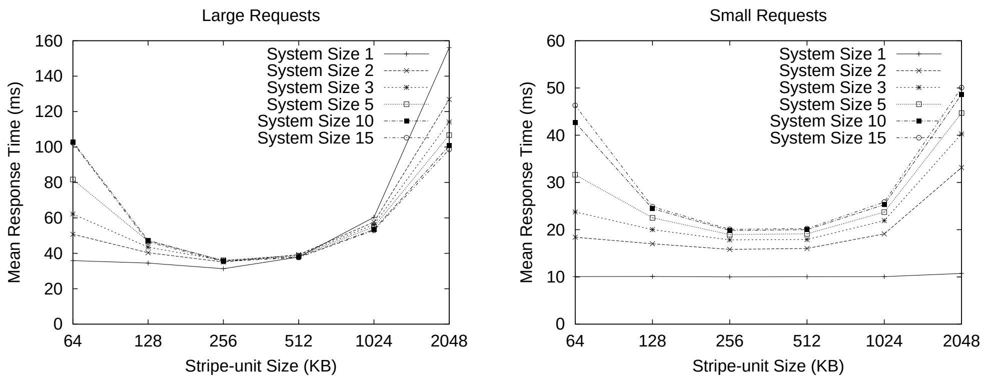


(a) Large Requests (b) Small Requests

Figure 4: Effect of system size for heterogeneous Poisson workloads. System size of 1 depicts narrow striping.

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 64KB to 2MB for large requests, and fix the stripe unit size for small requests at 4KB. Since the stripe-unit size of small requests did not have much impact on performance, we omit these results.

First, consider the impact of varying the large stripe unit size on large requests (see Figure 5(a)). When the large stripe unit is 64KB, a request of 1MB size causes an average of 16 blocks to be accessed per request. In case of narrow 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 1MB request accesses a larger number of disks, which reduces the number of sequential accesses on a disk and also increases the queue interference for both large and small requests. Consequently, narrow striping outperforms wide striping for a 64KB stripe unit size. As we increase the stripe unit size to 512 KB and beyond, the impact of loss in sequential access goes down. This coupled with the larger number of disk heads that are available for each request in wide striping leads to better performance for wide striping. Since stripe unit is not varied for small requests, it is impacted mainly by the utilization levels resulting from the different stripe unit choices for large requests (see Figure 5(b)). For small requests, due to the interference from large requests, wide striping leads to higher response time. Since disk overhead, and consequently utilization, is higher in wide striping at smaller stripe unit sizes, small requests see worse response times. As stripe unit size is increased the disk overhead decreases and hence the relative response time performance of wide striping improves. But, beyond 512 KB, the transfer times of the large stripe units becomes significant, and the re-



(a) Large Requests (b) Small Requests

Figure 5: Effect of varying the stripe unit size of large requests. System size of 1 depicts narrow striping.

sponse times of the small requests increases 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 (IA) times of requests. We first vary the IA times of large requests from 11ms to 20ms with the IA time of small requests fixed at 4ms (see Figure 6). We then vary the IA times of small requests from 2ms to 7ms with the IA time for large requests fixed at 17ms (see Figure 7). The various combinations of inter-arrival times and background loads result in utilizations ranging from 40% to 80%.

Figure 6(a) shows that, for large requests, wide striping outperforms narrow striping at high utilization levels and has slightly worse performance at low utilization levels. This is because, at higher 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 better load balancing in wide striping). Small requests, on the other hand, see uniformly worse performance due to the interference from large requests (see Figure 6(b)). The interference decreases at lower request rates and reduces the performance gap between the two systems.

The behavior is reversed when we vary the IA time of small requests (see Figure 7). At low inter-arrival times, large requests see maximum interference from small requests, and wide striping yields worse response times as a result. As the IA time is increased, the interference decreases, and the load balancing effect dominates leading to better response time in wide striping. For small requests, the response time difference in narrow and wide striping is always in the range of the transfer time for one stripe unit of a large request.

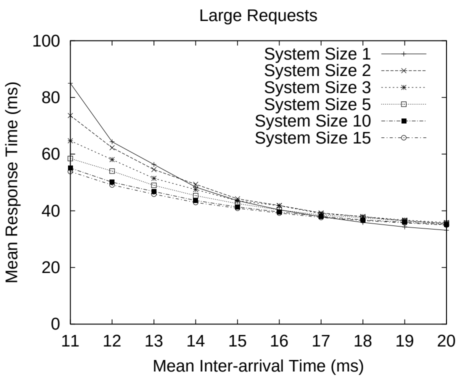
Effect of Request Size: In this experiment, we study the impact of the request size of large requests on the performance of wide and narrow striping. Varying the request size of small requests (in the range 2KB-16KB) did not have much impact, so we omit the results. We vary the average request size for large requests from 64KB to 2 MB (see Figure 8). The stripe unit size was chosen to be half the average request size for large requests; the average request size as well as the stripe unit size was fixed at 4 KB for small requests.

Figure 8(a) demonstrates that for large streams, initially (i.e., at small request sizes), queue interference results in slightly higher (approximately average seek time) response time in wide-striping. However, as the request size increases, the utilization increases and wide-striping leads to lower response times due to better load balancing. On the other hand, for small requests, wide-striping leads to larger response times, and the performance difference increases as we increase the large request size due to the increased transfer times of the large requests.

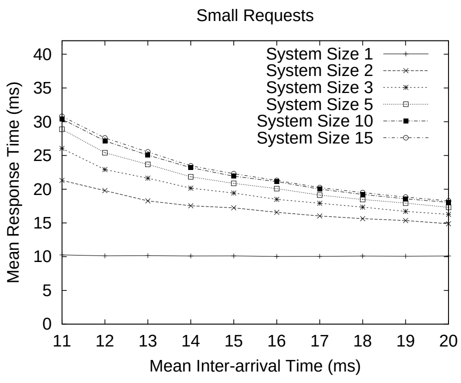
Effect of Writes: The above experiments have focused solely on read requests. In this experiment, we study the impact of write requests by varying the fraction of write requests in the workload. We vary the fraction of write requests from 10% to 90% and measure their impact on the response times in the wide and narrow striped systems. Recall that we simulate a write-back LRU cache.

We first vary the percentage of writes of the large requests with the small requests set to be read only (see Figure 9). Due to the write-back nature of the cache, all write requests return immediately after updating the cache. Consequently, the response times of write requests is identical in both narrow and wide striping. Hence, the overall response times (for both reads and writes) is governed mostly by read response times and the relative fraction of reads. In general, increasing the percentage of write requests increases the background load due to dirty cache flushes as well as the effective utilization (since the parity block also needs to be updated on a write[2]). Both of these factors interfere with read requests. For large requests, the increased interference is offset by the better load dispersion capability of wide striping, causing wide striping to outperform narrow striping—this performance advantage improves for larger system sizes (see Figure 9(a)). For small requests, on the other hand, the interference effect dominates at low utilization, causing wide striping to yield worse response times (see Figure 9(b)). As the percentage of writes is increased beyond 70%, wide striping outperforms narrow striping. This is because the interference from background

[2]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.

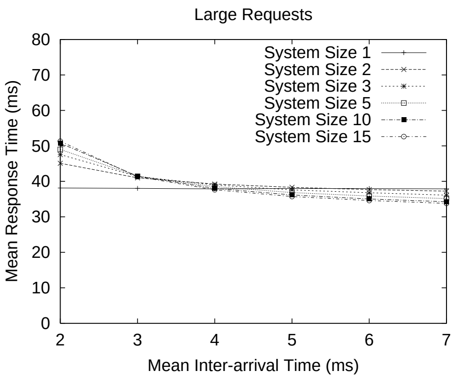


(a) Large Requests

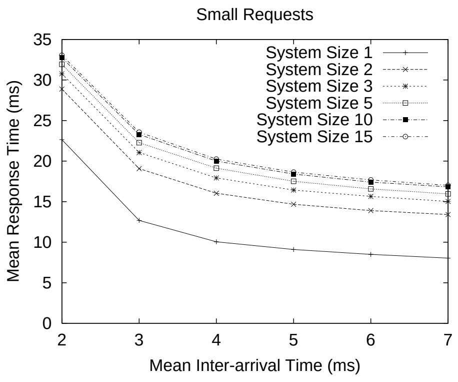


(b) Small Requests

Figure 6: Effect of the inter-arrival times of large requests. System size of 1 depicts narrow striping.



(a) Large Requests



(b) Small Requests

Figure 7: Effect of inter-arrival times of small requests. System size of 1 depicts narrow striping.

cache flushes and parity updates becomes dominant in write-intensive workloads, and wide striping yields better load balancing properties in the presence of such interference.

Next we vary the percentage of writes for the small requests (see Figure 10). The large request streams issue only read requests. As we increase the percentage of small write requests we see that large requests see queue interference in a wide-striped system; consequently narrow striping gives better performance. For small requests, as the write percentage is increased and the utilization goes up, the role of load balancing becomes significant and the performance of wide-striping improves, giving comparable performance at high write percentages.

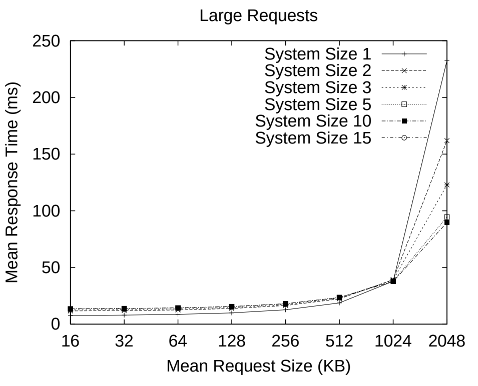
3.2.3 Summary

The above experiments compared a load-balanced, interference-free narrow striped system with a wide striped system using homogeneous and heterogeneous workloads. Our experiments demonstrate that, in the case of homogeneous workloads, narrow striping yields $10 - 15\%$ better response times for some scenarios, while the two systems yield comparable performance for most other scenarios. In case of heterogeneous workloads, our experiments demonstrated that if the stripe unit size is chosen appropriately, then wide striping yields *better* response times for large requests in *most* scenarios. In some cases, wide striping yields higher response times (in the range of an average seek time). For small requests, on the other hand, wide striping yields worse performance in most scenarios (the performance difference is in range of transfer time of a large stripe unit). In general, we find that as utilization increases (for instance, by increasing write percentage), wide-striping leads to better performance.

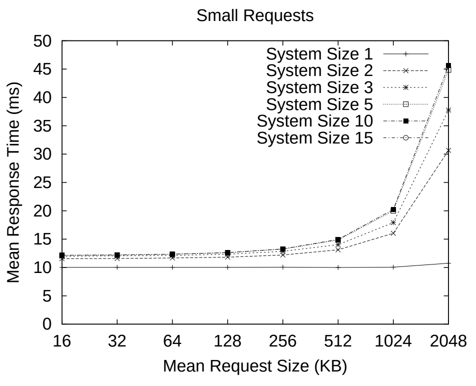
3.3 Impact of Inter-Stream Interference

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.

To introduce interference systematically 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 0 to 1 and measure its impact on the response times of large and small requests (correlation of 0

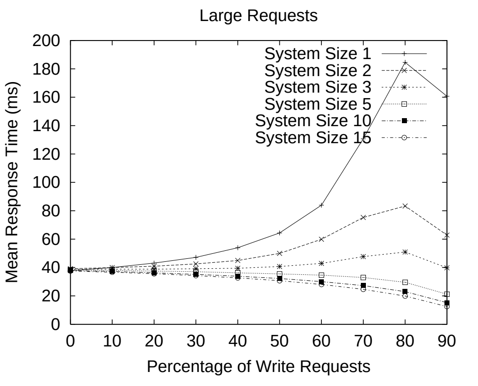


(a) Large Requests

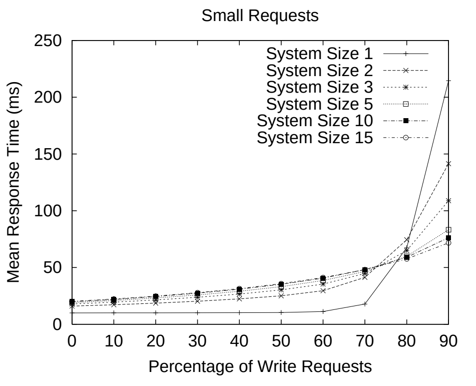


(b) Small Requests

Figure 8: Effect of request size of large requests. System size of 1 depicts narrow striping.



(a) Large Requests



(b) Small Requests

Figure 9: Effect of percentage of large write requests. System size of 1 depicts narrow striping.

implies that L_i and S_i are never on simultaneously while 1 implies that they are always on simultaneously). We control the correlation by varying the overlap fraction i.e., mean time for which the two streams are ON simultaneously. For simplicity we assume that the correlated streams have the same ON periods; also we assume the OFF period to have the same duration as the ON period. This gives us a high degree of control on stream correlations. For a correlation of x, $0 \leq x \leq 0.5$, the overlap fraction is uniformly distributed between *0* and *2*x*. For correlations between *0.5* and *1*, the overlap fraction is uniformly distributed between *2*x-1* and *1.0*.

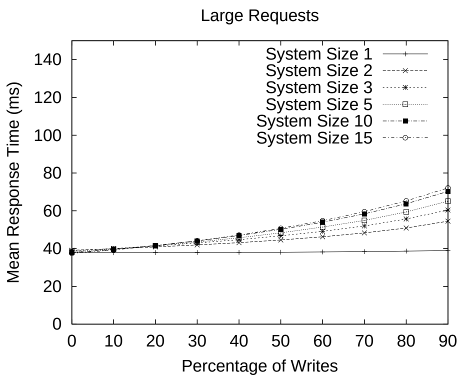
Figure 11 plots the impact of correlation on response time in narrow and wide striped systems. As the figure demonstrates, the performance of wide-striping improves with increase in correlation, with wide-striping performing better for both small and large request sizes for correlation values higher than 0.25. Observe that as correlation increases, the probability of temporary load-imbalance in the narrow striped system increases. Since wide-striping yields better load-balancing, it leads to better performance as correlation increases.

3.4 Impact of Load Skews: Trace-driven Simulations

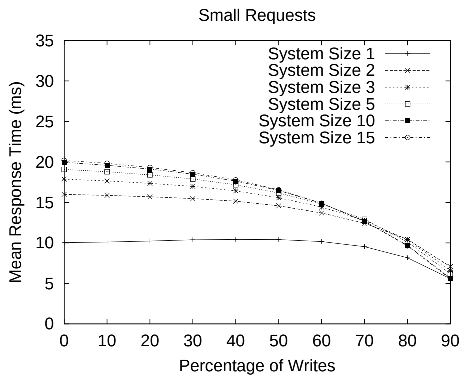
We use the trace workloads listed in Table 2 to evaluate the impact of load imbalance on the performance of narrow and wide striping. The traces have a mix of read and write I/Os and small and large I/Os. To illustrate, the OLTP-1 trace has a large fraction of small writes (mean request size is 2.5KB), 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 traces instead of a synthetic ON-OFF process. Due to the high percentage of writes in the OLTP streams, a cache of sufficient size, resulted in similar performance for both narrow and wide striping, when in the write back mode; so in the following, we have the cache in the write through mode.

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 further eliminate 4 streams from the OLTP traces as these were found to be capacity bound. We then partition the remaining 48 streams into four sets such that each set is load-balanced. We use the write-weighted average IOPS[3] 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

[3]I/O Operations Per Second

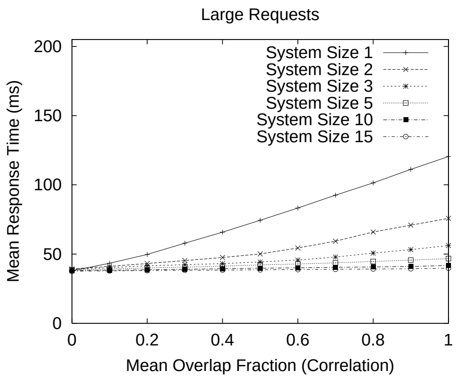


(a) Large Requests

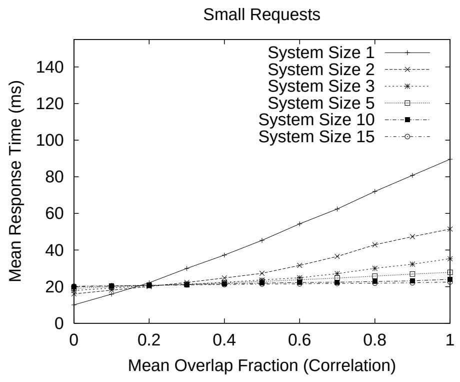


(b) Small Requests

Figure 10: Effect of percentage of small write requests. System size of 1 depicts narrow striping.



(a) Large Requests



(b) Small Requests

Figure 11: Impact of inter-stream interference. System size of 1 depicts narrow striping.

as a criteria for load balancing.

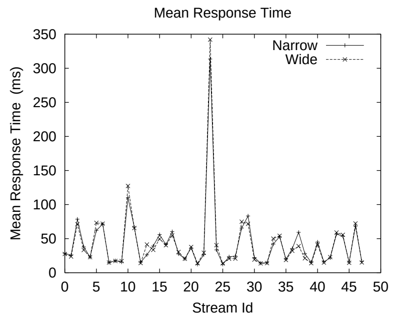
We employ a greedy algorithm for partitioning the streams. The algorithm creates a random permutation of the streams and assigns them to partitions one at a time, so that each stream is mapped to the partition that results in the least imbalance (the imbalance is defined to the difference between the 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 12 plots the average response time and the 95^{th} percentile of the response time, 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 that of a narrow striped system. Figure 12(c) shows the mean disk utilizations for the disks in the system (the X axis is the disk id). Observe that variance in the mean disk utilizations across the disks in the system is is lower in a wide-striped system due to better load balancing. Also, even for the case of narrow striping the variance in disk utilizations is low since the partitions are load balanced (a partition comprises of five consecutive disks).

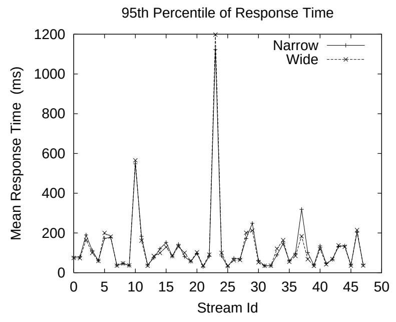
Next we introduce load imbalance across the partitions and compare the performance of narrow and wide striping. To introduce imbalance we simply scale the inter-arrival times for all streams on the first two partitions by a factor of 0.75 (streams 0-25). In the narrow striped system this increases the load on the first two partitions (disks 0-9) and the load on the other two partitions remains unchanged; for a wide striped system however the load across all the partitions goes up. Figure 13 plots the results. As can be seen in Figure 13 (c) the mean utilization across the first two partitions (first ten disks) has gone up in narrow striping, and the utilization across all the partitions had gone up for wide striping (compare with Figure 12 (c)). A look at the plots for the average response time as well as the plot for the 95^{th} percentile of the response time shows that wide striping outperforms narrow striping for streams on the first two partitions, and their performance on the remaining two partitions is similar. Thus we see that, due to better load balancing, wide striping outperforms narrow striping in the presence of load imbalances.

3.5 Experiments on a Storage System Testbed

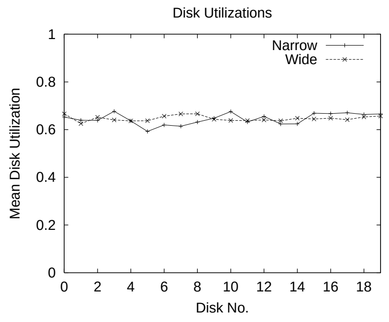
In this section, we compare the performance of narrow and wide striping on our storage testbed using a synthetic workload and two database benchmark workloads—TPC-C and TPC-H. system. For the synthetic and TPC-H workloads, we



(a) Mean Response Time

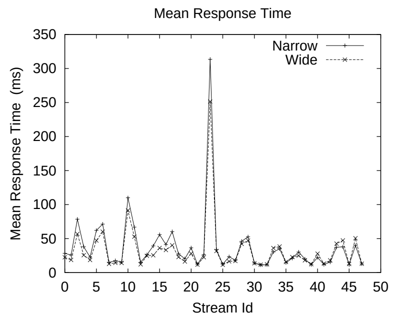


(b) 95^{th} Percentile of the Response Time

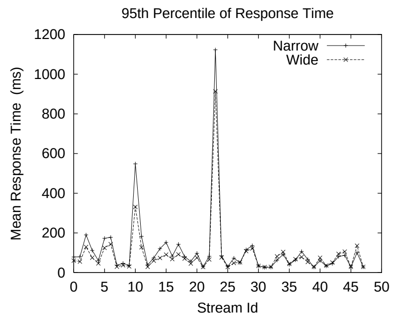


(c) Mean Disk Utilizations

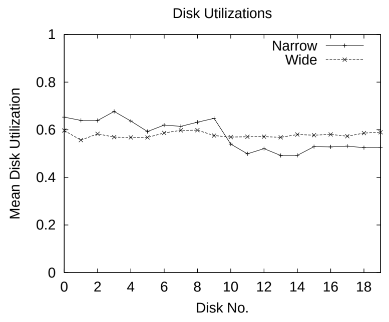
Figure 12: Trace Driven Simulations



(a) Mean Response Time



(b) 95^{th} Percentile of the Response Time



(c) Mean Disk Utilizations

Figure 13: Trace Driven Simulations with Load Imbalance

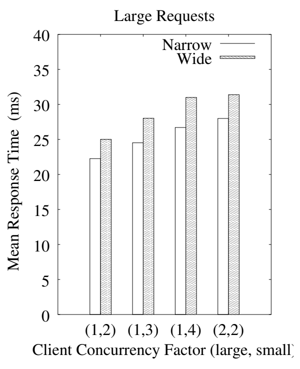
use a FAStT-700 storage subsystem, and for the TPC-C workload, we use a SSA-based RAID subsystem.

3.5.1 Synthetic Workload

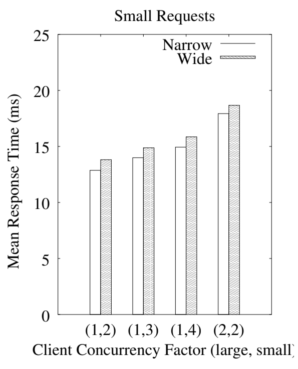
The workload consists of two closed-loop streams, one large and one small, accessing two independent stores on a RAID-5 array simultaneously. Each store was of size 2GB and was created on a $4 + p$ RAID-5 array on FAStT. For large requests the stripe unit size of the store was 256 KB and for small requests it was configured to be 8 KB. We used the *Linux Logical Volume Manager* (LVM) for wide striping the stores. The mean request size for large and small requests was chosen to be 512 KB and 8 KB, respectively. Figure 14 shows the response time performance of narrow and wide-striping, for various combinations of concurrency factors of the clients accessing the large and small stores, respectively. As the experiments demonstrate, the performance of wide striping is within $10 - 15\%$ of the narrow striped system.

3.5.2 TPC-H Workload

TPC-H is a decision support benchmark. It was used in [2] to illustrate the benefit of narrow striping. We use a setup similar to the one in [2] with IBM DB2 UDB instead of MS SQL server. We setup the TPC-H database on a 1.6 GHz Pentium 4 with 512 MB RAM running Linux 2.4.18. This was connected to the FAStT-700 storage system using Fibre Channel. The page size and the extent size for the database were chosen to be 4 KB and 32 KB, respectively. The scale factor for the database was set to 1 (a 1 GB database).



(a) Large Requests



(b) Small Requests

Figure 14: Heterogeneous Workload: Closed-loop Testbed Experiments

For narrow striping, we used the placement described in [2]. The table *lineitem* was spread uniformly over 5 disk drives, *orders* was spread uniformly over three other disk drives, and all the other tables and indexes (including the indexes for the tables *lineitem* and *order*) were placed in a third logical volume (called *rest*) which was striped across all the 8 disk drives. In the wide-striped case the tables *lineitem* and *orders* were striped across all the 8 disk drives, as was the logical volume *rest*. In both cases the system temporary tables were placed on a ninth disk drive, also on the FAStT-700. The stripe unit size was chosen to be 32 KB in all cases.

Figure 15(a) shows the query execution times for narrow and wide striping for a single stream run (power run) of TPC-H. Since this is an unaudited run; the query execution times are normalized. As the figure demonstrates, most of the queries have similar execution times. Only for queries 20 and 21 do we see a $5-10\%$ performance difference; narrow striping outperforms wide striping for query 20, and vice versa for query 21.

Figures 15(b) and 15(c) plot the I/O profile for the *lineitem*, *orders*, and *rest* volumes for narrow and wide striping, respectively. The figure demonstrates that the I/O profile are indeed very similar in both cases. It also demonstrates that lineitem and orders are indeed the two important tables and the narrow placement algorithm suggested in [2] appear to be valid for DB2 as well. Overall, we find that when the tables are carefully mapped to arrays in narrow striping, the two systems perform comparably (note that, no placement optimizations are necessary for wide striping).

3.5.3 TPC-C Workload

Our final experiment involves a comparison of narrow and wide striping using TPC-C workload. Our testbed consists of a four-processor IBM RS6000 machine with 512 MB RAM and AIX 4.3.3. The machine contains 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 [4]). 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).

Striping	Sequential I/O Size	Sequential Throughput	Normalized TPC-C Throughput
Narrow	896 KB	25.43 MB/s	N TpmC
Wide	896 KB	20.09 MB/s	1.33 N TpmC
Narrow	7 MB	29.45 MB/s	N TpmC
Wide	7 MB	36.86 MB/s	0.82 N TpmC

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

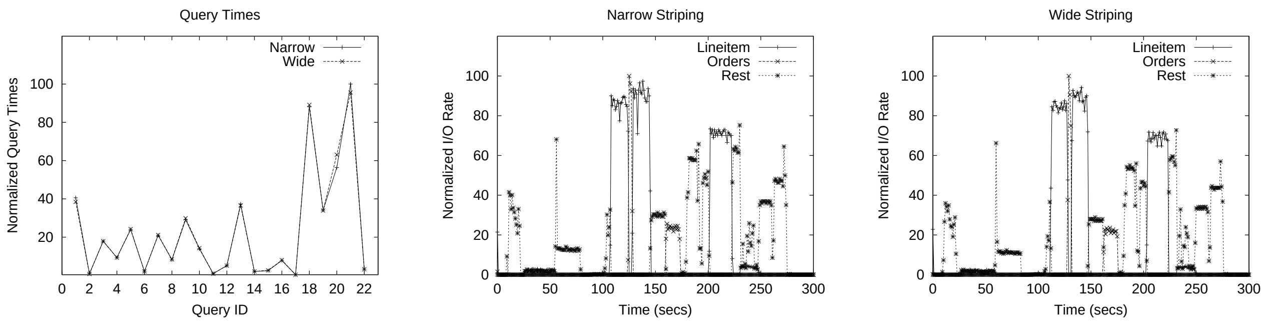
We first experiment with a narrow striped system by running the TPC-C benchmark on one array and the sequential application on the other array. We find the TPC-C throughput to be N TpmC (the exact number withheld since this is an unaudited run), while the throughput of the sequential application is 25.43 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 1.33N TpmC when the sequential workload uses 896 KB requests and is 0.82N TpmC for 7MB requests. The corresponding sequential workload throughput is 20.09 MB/s and 36.86 Mb/s, respectively.

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 for small requests when the streams can be ideally



(a) Query Times (b) Narrow Striping: I/O Profile (c) Wide Striping: I/O Profile

Figure 15: Comparison using the TPC-H Benchmark

partitioned such that the partitions are load-balanced and there is very little interference between streams within a partition. However, in the presence of workload skews that occur in real I/O workloads wide striping outperforms narrow striping. In our trace-driven experiments, we found that when the the average load was balanced, wide-striping performed comparably to narrow-striping. However, when we introduced load imbalance by increasing the load on some partitions, wide striping outperformed narrow striping for the streams on the heavily loaded partitions while performing comparably for the remaining streams. 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 our closed-loop testbed experiments and the TPC-H experiments we found the performance of narrow and wide striping to be comparable.

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 [5]. This entails moving stores between arrays to achieve a new layout [14]. 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 for dynamically changing (non-stationary) workloads. Policy-managed systems that employ narrow striping [4, 5] 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, 16].

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 QoS-aware disk scheduling mechanisms exist in the storage system [17], 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.

A final issue is system reliability. In narrow striping, when multiple disks fails on a RAID array, only stores mapped onto that array are rendered unavailable. In contrast, all stores are impacted by the failure of any one RAID array in wide striping. The overall choice between wide and narrow striping will be dictated by a combination of the above factors.

4 Related Work

The design of policy-managed storage systems was pioneered by [4, 6, 5, 14], 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 access correlation between streams, and collocates bandwidth-bound and space-bound objects 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, 12, 16]. 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, at least one database vendor has recently advocated wide striping due to its inherent simplicity [13]. A cursory evaluation of wide striping combined with mirroring, 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.

5 Concluding Remarks

Storage management cost is a significant fraction of the total cost of ownership of large database applications. 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 for database workloads. 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 simplicity and performance. Using synthetic and real I/O workloads, we found that an idealized narrow striped system can outperform a comparable wide-striped system for small requests. However, wide striping outperforms narrow striped systems in the presence of workload skews that occur in real systems; the two systems perform comparably for a variety of other real-world scenarios. 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, and (ii) it is feasible to use data migration to handle workload skews and workload interference. In general, we argue for simplicity and 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] S. Agrawal, S. Chaudhuri, A. Das, and V. Narasayya. Automating layout of relational databases. In *Proceedings of the 19th International Conference on Data Engineering, Bangalore, India*, 2003.

[3] N. Allen. Don't waste your storage dollars. Research Report, Gartner Group, March 2001.

[4] 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*, 2002.

[5] 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.

[6] 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.

[7] 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.

[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] E. Lamb. Hardware spending matters. *Red Herring*, pages 32–22, June 2001.

[12] 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.

[13] J. Loaiza. Optimal storage configuration made easy. Technical report, Oracle Corporation. Available from http://otn.oracle.com/deploy/performance/pdf/opt_storage_conf.pdf.

[14] 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.

[15] 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.

[16] P. Scheuermann, G. Weikum, and P. Zabback. Data partitioning and load balancing in parallel disk systems. *VLDB Journal*, 7(1):48–66, 1998.

[17] P Shenoy and H M. Vin. Cello: A disk scheduling framework for next generation operating systems. In *Proceedings of ACM SIGMETRICS Conference, Madison, WI*, pages 44–55, June 1998.

[18] 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.

A Comparison using Homogeneous Workloads

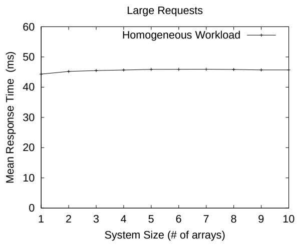
In this appendix we present the detailed results of our homogeneous workload simulations experiments. We experiment with large requests that have a mean request size of 1 MB and a stripe unit size of 512KB. We repeat each experiment with small requests that have a mean size of 4KB and a stripe unit size of 4KB. Unless specified otherwise, we choose request rates that yield a utilization of around 60-65%; this corresponds to a mean inter-arrival time of 17 ms for large requests and 4 ms for small requests, respectively.

Effect of System Size: We vary the number of arrays in the system from 1 to 10 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. Figure 16 plots the results.

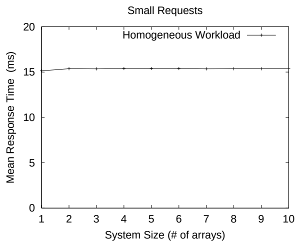
The figure shows that the performance of the two systems is similar over a range of system sizes for both large and small requests. Increasing the system size results in interference between streams in wide striping since all stores span all arrays. However, since all stores span all arrays, this also leads to better load balancing across arrays. As we increase the system size, the benefits of load balancing balance the impact of interference, and the response times remain almost unchanged.

Effect of Stripe Unit Size: In this experiment we study the impact of changing the stripe unit size. Varying the stripe unit size of small requests did not have much impact, so we omit the results. The stripe unit size of large requests was varied from 128 KB to 2 MB. The average request size of the large requests was kept fixed at 1 MB. Figure 17 plots the results.

For large requests, when the stripe unit size is smaller as compared to the average request size, wide-striping gives higher response times as compared to narrow striping. This is because, although a smaller stripe unit size results in increased parallelism, it also increases the sequentiality breakdown and the probability of interference with requests from streams accessing other stores. To wit, an average request size of 1 MB would result in 8 disk accesses for a stripe unit size of 128 KB, as compared to 2 disk accesses for a stripe unit size of 512 KB. The sequentiality of access is maintained in narrow-striping since all requests for a store access the same array. An increase in the stripe unit size reduces the extent of sequentiality breakdown, and narrow and wide striping give comparable performance.



(a) Large Requests



(b) Small Requests

Figure 16: Homogeneous Workload: Effect of System Size

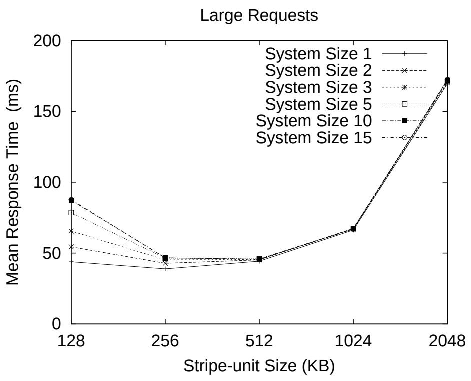


Figure 17: Homogeneous Workload: Effect of Stripe-unit Size

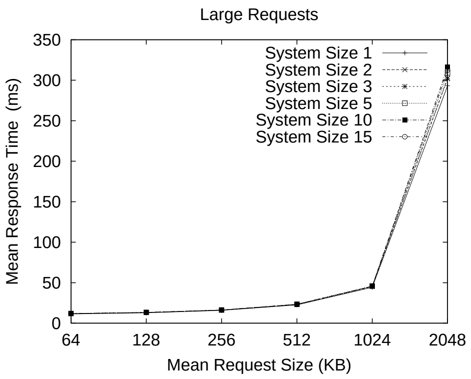


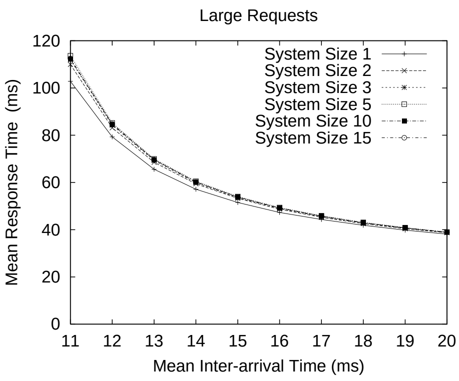
Figure 19: Homogeneous Workload: Effect of Request Size

Effect of Utilization Level: In this experiment, we study the impact of utilization level by varying the mean inter-arrival times (IA) of requests. The IA time for large (small) requests is varied from 14 ms to 20 ms (3ms to 7ms) in steps of 1 ms. Figure 18 shows the results for the large and the small case, respectively.

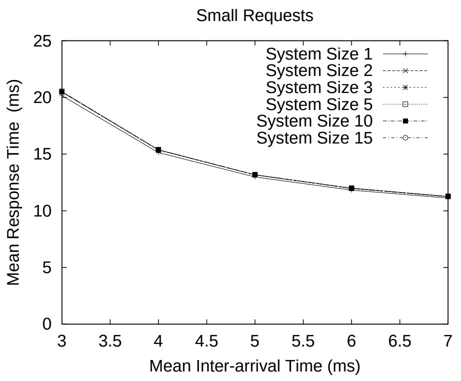
Figure 18 (a) shows that for large requests, as one decreases the IA times the relative performance of narrow striping improves slightly. This is because, at low IA times the request rate is higher, and streams see increased interference from other streams in wide striping. For larger IA times narrow and wide striping give comparable performance. Varying the IA times for smaller requests results in similar behavior (see Figure 18 (b)); the difference in response times between narrow and wide striping in this case however, are smaller than that observed for large requests, because of the smaller transfer time of small requests.

Effect of Request Size: Next we study the effect of changing the request size. Varying the request size of small requests did not have much impact so we omit the results. The request size of large requests is varied from 64 KB to 128 KB. The stripe unit size was chosen to be half the average request size. Figure 19 plots the results. Narrow and wide striping give similar performance for most request sizes. For very large request sizes (2 MB), the interference between request streams results in wide striping giving slightly larger response times.

Effect of Percentage of Writes: In this experiment we

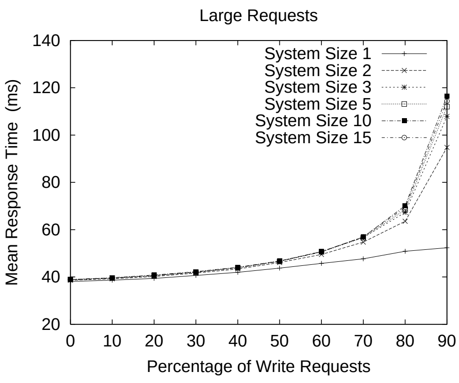


(a) Large Requests

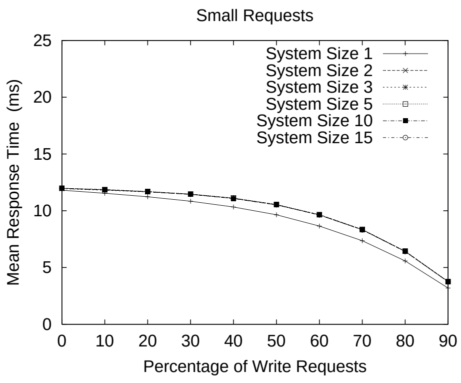


(b) Small Requests

Figure 18: Homogeneous Workload: Effect of Utilization Level



(a) Large Requests



(b) Small Requests

Figure 20: Homogeneous Workload: Effect of Percentage of Writes

study the effect of varying the percentage of writes. The percentage of writes was varied from 0 % to 90 %. We chose inter-arrival times of 20 ms and 6 ms for large and small requests, respectively. Figure 20 plots the results.

For large requests (see Figure 20 (a)) we observe that as we increase the percentage of writes the performance difference between narrow and wide striping increases, with wide-striping giving higher response times. This is because, increasing the percentage of writes, increases the background load due to dirty cache flushes, which increases the interference seen by request streams in wide striping. Small requests (see Figure 20 (b)) observe similar behavior; the impact of interference from background load due to dirty cache flushes however, is less pronounced, due to the smaller size of requests.