Architectural Considerations for Next Generation File Systems

Prashant Shenoy, **Pawan Goyal**† and **Harrick M. Vin**‡

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

†AT&T Research
180 Park Avenue
Florham Park, NJ 07932
goyal@research.att.com

‡Department of Computer Sciences,
University of Texas
Austin, TX 78712
vin@cs.utexas.edu

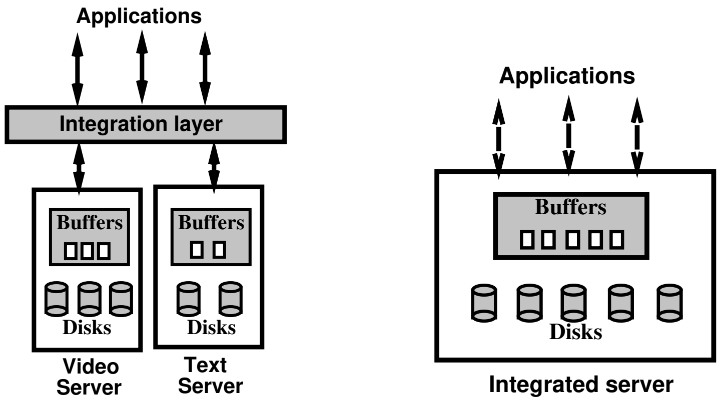
Abstract

Integration—supporting multiple application classes with heterogeneous requirements—is an emerging trend in networks, file systems, and operating systems. In this paper, we evaluate two architectural alternatives—partitioned and integrated—for designing next generation file systems. Whereas a partitioned server employs a separate component file system for each application class, an integrated file server shares its resources across all application classes. We evaluate the performance of these two architectures with respect to sharing of disk bandwidth. We show that although the problem of sharing disk bandwidth in integrated file systems is conceptually similar to that of sharing network link bandwidth in integrated services networks, the arguments that demonstrate the superiority of integrated services networks are not applicable to file systems. Furthermore, we show that: (i) an integrated server outperforms the partitioned server in a large operating region and has slightly worse performance in the remaining region, (ii) the capacity of an integrated server is larger than that of the partitioned server, and (iii) an integrated server outperforms the partitioned server by up to a factor of 6 in the presence of bursty workloads.

1 Introduction

1.1 Motivation

The current euphoria over the World Wide Web does not do full justice to the potential of the Internet; with the manifold increases in CPU processing power and network bandwidth, it is inevitable that the Internet will support distributed applications that process massive amounts of data for visualization and support real-time interactivity. For instance, a digital library of satellite imagery might be processed for feature extraction or visualization; a virtual environment for training fire fighters will involve distributed simulations and real-time user interactivity. These applications differ from conventional applications in at least two ways. First, they involve storing, transporting, and processing heterogeneous data types (e.g., images, audio, video, text, etc.). Second, they impose diverse performance requirements (e.g., with respect to timeliness) on networks and operating systems. Traditionally, such heterogeneous applications have been supported in isolated and dedicated environments. Recently, triggered by the technological advances and the proliferation of the Internet, the problem of supporting these applications in general-purpose computing, communication, and storage environments—where resources are shared among several, potentially conflicting, applications—is receiving much attention.



(a) Partitioned file server (b) Integrated file server

Figure 1: Partitioned and integrated file servers supporting text and video applications. The partitioned architecture divides the server resources among multiple component file systems, and employs an integration layer that provides a uniform mechanism to access files. The integrated architecture employs a single server that multiplexes all the resources among multiple application classes.

In this paper, we focus on the design of next-generation *file systems* that, unlike conventional file systems, export multiple service classes and simultaneously support heterogeneous data types and applications [16, 19]. We consider two architectural alternatives for designing such file systems (see Figure 1):

- A *partitioned* architecture that: (1) divides the server resources among multiple component file systems, each optimized for a particular application class or data type, and (2) employs an integration layer that provides a uniform mechanism to access files managed by separate file systems.
- An *integrated* architecture that multiplexes all the resources available at a server—the storage space, the disk bandwidth, and the buffer cache—among multiple application classes and data types.

Since techniques for designing file systems that are optimized for a single application class are well-understood [9, 13, 22, 23], partitioned file systems are easy to design and implement. The design of integrated file systems, on the other hand, is challenging due to the heterogeneous performance requirements of data types and applications. However, since such file systems share all resources among all application classes, they can *potentially* provide better performance to applications as compared to a partitioned system. This hypothesis has been the basis for the design of several integrated file systems [1, 12, 19]. Most of these efforts have focused on the design and analysis of novel techniques for managing resources in integrated servers. Proponents of the partitioned architecture argue that the gains due to integration are insignificant to warrant its increased complexity. Most papers on the design of integrated file systems, on the other hand, implicitly assume that such an architecture is, in fact, necessary and give little justification for their approach. Surprisingly, there has been no study that systematically evaluates the two architectures to quantify these tradeoffs. In this paper, we focus on evaluating the tradeoffs between the two architectural alternatives for designing next-generation file systems. Rather than advocating a particular design alternative, our goal here is to provide a systematic evaluation—with respect to application performance—of the two architectural choices. We believe that such an evaluation will provide valuable insights and guidelines to future file system designers.

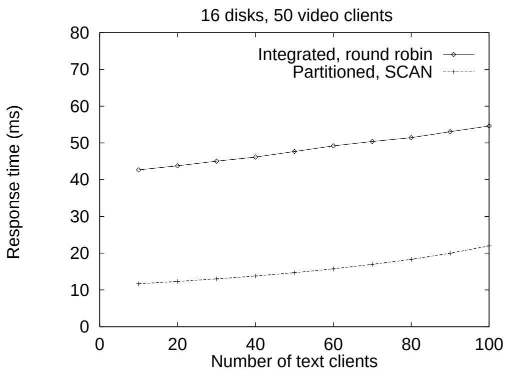


Figure 2: Limitation of fair scheduling algorithms for integrated file servers. Use of a round-robin scheduling algorithm to service text and video requests in integrated file system causes the disk to incur a large seek and rotational latency overhead when switching between requests. Hence, the response time is significantly worse than a partitioned system that employs separate servers for text and video.

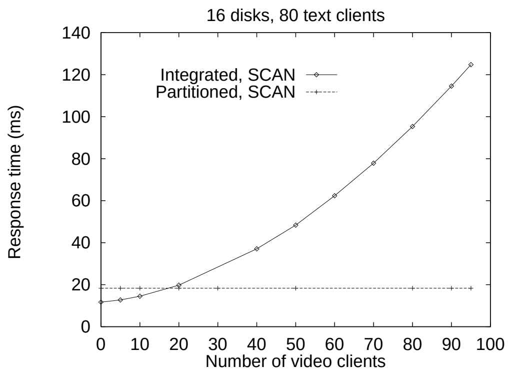
1.2 Problem Formulation

A file system manages several resources such as disk bandwidth, storage space, and buffer cache. An integrated file system can benefit from dynamic multiplexing of each of these resources. Since application performance is most affected by the availability of disk bandwidth, in this paper, we restrict our evaluation to the potential performance gains that can be achieved by sharing disk bandwidth.

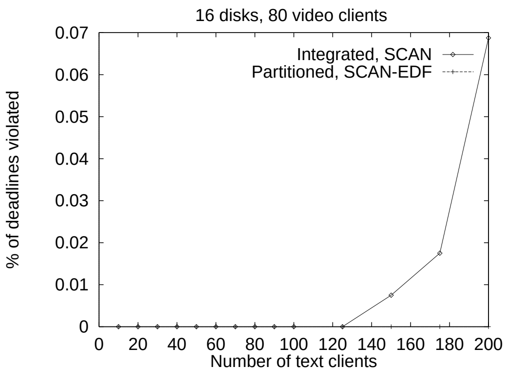
The issues that arise in sharing of disk bandwidth in an integrated file system are conceptually similar to those that arise in multiplexing link bandwidth in integrated services networks. An integrated services network, in addition to providing efficiency and economy to network providers and convenience to users, also provides better performance to various applications that use it. To illustrate, consider an environment with two separate networks—each with capacity C—serving different application classes, and an integrated services network with capacity $2C$. If we assume that both networks carry packets of the same size then the integrated services network can provide to applications *no worse, and often significantly better*, performance than the separate networks by simply employing a round-robin scheduler for packets belonging to the two application classes (see [2] for an example of an algorithm when packets are of unequal size). This is because, when both the application classes have packets to transmit, they each receive C units of bandwidth—similar to the separate network scenario; but when one of the application classes does not utilize its fair share, the idle network bandwidth is used to provide better performance to the other application class. The increased efficiency due to *statistical sharing* of network bandwidth is the central design principle of integrated services networks. In fact, it has been argued that the *least* efficient network design is the one that uses separate networks, each optimized for a different application class [17].

This leads to the following fundamental question: are the arguments that demonstrate the superiority of an integrated services network over separate networks also applicable when comparing the performance of partitioned and integrated file servers? Surprisingly, the answer to this question is not straightforward. This is because:

1. There is a subtle difference in the characteristics of network links and disks: *network link throughput is unaffected by the relative order of servicing requests; but for disks, the relative order of servicing requests governs the overall disk throughput*. Round-robin and fair scheduling algorithms (e.g., WFQ [6], WF^{2}Q [2])



(a) Text response time



(b) Percentage of deadline violations for video

Figure 3: Limitation of SCAN disk scheduling algorithm for integrated file systems. Figure (a) shows that, the presence of video requests interferes with text requests in the integrated server and degrades text response time. Figure (b) shows that text requests interfere with video requests and cause deadlines of the latter to be violated. Text and video requests access mutually exclusive set of disks in the partitioned server; hence, the presence of video requests has no effect on text requests and vice versa.

determine the order for servicing requests solely on the basis of the fairness criterion; they ignore the seek time and rotational latency incurred while servicing each disk access request. Consequently, using them to arbitrate access to disk bandwidth yields poor performance. In fact, the overhead incurred by such algorithms may more than offset the statistical multiplexing gains obtained by the integrated architecture (see Figure 2).

2. Conventional disk scheduling algorithms such as SCAN and SATF (Shortest Access Time First) [3, 4, 7, 8, 10, 21, 25, 26] determine the order of servicing disk requests solely on the basis of the relative positions of the blocks to be accessed on disk, and hence minimize the seek time and rotational latency overhead incurred while servicing requests. However, they do not provide any isolation across classes: a burst of request arrival for best-effort application may violate the deadlines of requests from real-time applications, and the arrival of a large number of real-time requests can cause response times for best-effort applications to degrade (see Figure 3).

To realize the benefits of statistical multiplexing when applications have diverse requirements, it has been argued that a disk scheduling algorithm should: (1) align the service it provides with the application needs, (2) protect application classes from one another, (3) be work-conserving and adapt to changes in work-load, (4) minimize the seek time and rotational latency overhead incurred during access, and (5) be computationally efficient. In the recent past, several algorithms have been proposed to address these issues [14, 18, 24]. These algorithms are heuristics for meeting these requirements and unlike packet scheduling algorithms do not formally guarantee these properties. Hence, unlike integrated services networks, the superiority of integrated file servers from the perspective of application performance can not be theoretically argued and formally demonstrated. Therefore, we experimentally answer the following question: given an appropriate disk scheduling algorithm, can an integrated file server achieve statistical multiplexing gains similar to those in integrated services networks? To formulate the problem more precisely, consider the following special case of this general question.

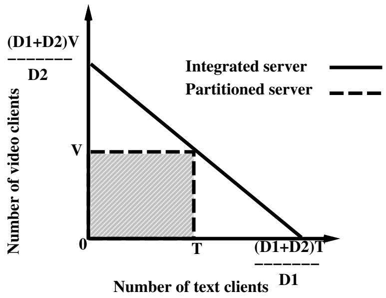


Figure 4: Boundaries of operation for a partitioned and an *ideal* integrated file server. A partitioned file server that partitions resources among a text and a video server can support up to T text and V video clients. An ideal integrated file server with an identical configuration can support up to $\frac{D_1+D_2}{D_2} * V$ video and $\frac{D_1+D_2}{D_1} * T$ text clients. Moreover, it provides comparable or better performance in the shaded region when the number of text and video clients is less than T and V.

Consider a file system that is required to support *text* and *video* applications. Let the file system be required to support T text and V video clients simultaneously. For the partitioned architecture, let D_1 and D_2, respectively, denote the number of disks required for the text and the video servers to meet this requirement. Now, consider an integrated server that multiplexes $(D_1 + D_2)$ disks among both application classes. In the ideal scenario (i.e., similar to the integrated services networks), we would expect the integrated server to provide *no worse and often better* performance when the number of text and video clients is less than T and V, respectively. Furthermore, we would expect it to support: (1) at least T text and V video clients simultaneously; (2) $\frac{D_1+D_2}{D_2} * V$ video clients when there are no text clients; and (3) $\frac{D_1+D_2}{D_1} * T$ text clients in the absence of any video clients. Finally, the capacity of the server should scale linearly between these two extremes. Figure 4 depicts this ideal behavior. In this scenario, we are interested in answering two questions:

1. Does the integrated architecture yield better performance in the shaded region in Figure 4 as is expected in an ideal server? Is it possible to achieve the ideal behavior depicted by the non-shaded region in Figure 4? That is, can the capacity of the integrated server scale linearly between $\frac{D_1+D_2}{D_2} * V$ video and $\frac{D_1+D_2}{D_1} * T$ text clients?

2. What is the expected difference in performance of the partitioned and integrated architectures in the presence of real-life (bursty) workloads?

1.3 Research Contributions of This Paper

In this paper, we evaluate the performance of the integrated and partitioned servers for text and video applications. Whereas text applications desire low average response time, video applications require that their deadlines be met. For these application classes, we show that an integrated server: (i) yields *better performance* than its counterpart over a large operating region but has slightly worse performance in the remaining region, (ii) has a *larger capacity* since it can support a larger number of clients from a class when the other class does not use its fair share, and (iii) is *self-adapting* since larger capacity and sharing of disk bandwidth enable it to better handle bursty loads that cause transient overloads.

We first compare the performance of the two architectures when the load is less than the maximum that can be supported by the partitioned system (the shaded region in Figure 4). Our experiments demonstrate that the integrated server outperforms the partitioned server by a significant amount in a majority of this region. Unlike an integrated services network, however, the integrated server yields slightly worse performance than its counterpart in a small region. Specifically, text requests see a worse response time at low text and heavy video loads (upper left hand corner of the region), whereas video request see a larger number of deadline violations at heavy text and video loads (upper right hand corner of the region). For video clients, we show that, in spite of the potential interference from text requests, the integrated server is able to provide matching performance to video requests (by meeting a comparable number of request deadlines).

Next, we determine if the capacity of an integrated server scales up to $\frac{D_1+D_2}{D_2} * V$ video and $\frac{D_1+D_2}{D_1} * T$ text clients. We demonstrate that the capacity curve of an integrated server is indeed similar to that of the ideal scenario; hence an integrated server can support a larger number of clients from an application class when the other class does not use its fair share. However, due to load imbalances in the system and the idiosyncrasies of the disk scheduling algorithm, there is a small (about 5%) degradation in the number of text and video clients supported as compared to the ideal server.

Finally, we demonstrate that the ability of an integrated server to support a larger number of clients enable it to adapt to bursty workloads that can cause transient overloads. Specifically, we show that, when the server is operating at 50% utilization, in the presence of bursty text loads, the average response time yielded by the integrated server is smaller by a factor of 6 as compared to the partitioned server. On an average, the integrated server yields a 40%–80% improvement in response time over its counterpart. Moreover, such bursty loads have little or no effect on the performance of video clients.

The rest of the paper is organized as follows. Section 2 describes our experimental methodology for comparing the partitioned and integrated architectures. Section 3 discusses our experimental results. Finally, Section 4 summarizes our results and highlights our key observations.

2 Experimental Methodology

To evaluate the tradeoffs between the two architectural alternatives for designing next-generation file systems, we conducted extensive simulations. In what follows, we describe our simulation environment, the metrics for our evaluation, and the workload generator.

2.1 Simulation Environment

We have implemented an event-based simulator to evaluate the relative performance of the partitioned and the integrated architectures. In this paper, we report the results obtained from the evaluation of partitioned and integrated servers that (1) support storage and retrieval of two data types—text and video, and (2) support two service classes—interactive best-effort and real-time.

To support the two application classes, the partitioned file system employs a text server and a video server. Each server manages a mutually exclusive set of disks; let D_1 and D_2, respectively, denote the number of disks managed by the text and the video server. Each file is striped across all the disks within a server. The text and video servers, respectively, provide interactive best-effort and real-time service to applications. The text server uses the SCAN disk scheduling algorithm [21] to service requests, while the video server uses SCAN-EDF [15]. Whereas SCAN services requests in the increasing order of cylinder numbers so as to reduce the seek time overheads, SCAN-EDF services requests in the increasing order of deadlines; requests with identical deadlines are serviced in SCAN order.

The integrated server multiplexes (D_1+D_2) disks among the two data types and application classes. Conventional scheduling algorithms such as SCAN are inadequate for such servers; recently, several disk scheduling algorithms that can support diverse application classes have been proposed [14, 18, 24]. Our simulator for the integrated server employs one such disk scheduling algorithm, namely *Cello* [18]. Cello allocates disk bandwidth to application classes at two time-scales. At a coarse time-scale, it determines the number of requests from each application class to be serviced, and at a fine time-scale, it determines the order for servicing the set of selected requests. Whereas the former enables Cello to protect application classes from one another and to adapt disk bandwidth allocation with changing work-load, the latter enables it to align the service provided with the application requirements while minimizing the seek time and rotational latency overhead. Additionally, Cello exploits characteristics of requests to align the service provided with application needs. For the two application classes under consideration, Cello delays real-time requests until their deadlines and uses the available slack to service interactive best-effort requests; this enables it to provide low average response times to the interactive best-effort applications without violating the deadlines of the real-time applications. Furthermore, it assigns weights to each application class and allocates bandwidth to classes in proportion to their weight; bandwidth unused by a class is reassigned to other classes with pending requests [18].

To derive the results presented in this paper, we configure the text and the video servers in the partitioned architecture with 8 disks each (i.e., $D_1 = D_2 = 8$), and the integrated server with 16 disks. We use stripe unit sizes of 8KB and 64KB, respectively, to stripe text and video files on the disk arrays. The text server employs a 64MB LRU buffer cache; the video server does not use caching. The integrated server also uses a 64MB LRU buffer cache for text requests. To ensure that the integrated server allocates equal bandwidth to each application class (and thereby mimic $D_1 = D_2 = 8$ in the partitioned architecture), we assign equal weights to the two application classes in the Cello scheduler (i.e., $w_1 : w_2 = 1 : 1$) [18]. We parameterize the disk simulator with the characteristics of the Seagate Elite3 disk (see Table 1). The seek time, rotational latency, and transfer times are computed using an empirically derived disk model [11]. The disk model in our simulator was validated using a real disk for the SCAN disk scheduling algorithm (since algorithms such as Cello and SCAN-EDF haven't been deployed in real systems, we were unable to validate the simulator for these algorithms).

2.2 Performance Metrics

Text applications are interactive best-effort in nature, and desire low average response times. Video applications, on the other hand, require request deadlines to be met. Consequently, to compare the performance of partitioned and integrated servers, we choose the *average response time* of text retrieval requests as the metric for text applications, and the *percentage of deadline violations* as the metric for video applications. We measure the capacities of the two architectures in terms of the maximum number of text and video clients that can be supported such that the average response times and the percentage of deadline violations are smaller than thresholds τ_t and τ_v, respectively.

Table 1: Disk parameters of Seagate Elite3 disk

Disk capacity	2 GBytes	Minimum seek time	1.7 ms
Bytes per sector	512 KB	Maximum seek time	22.5 ms
Sector per track	99	Maximum rotational latency	11.1 ms
Tracks per cylinder	21	Average seek time	11.0
Cylinders per disk	2627	Average Transfer rate	4.6 MB/s

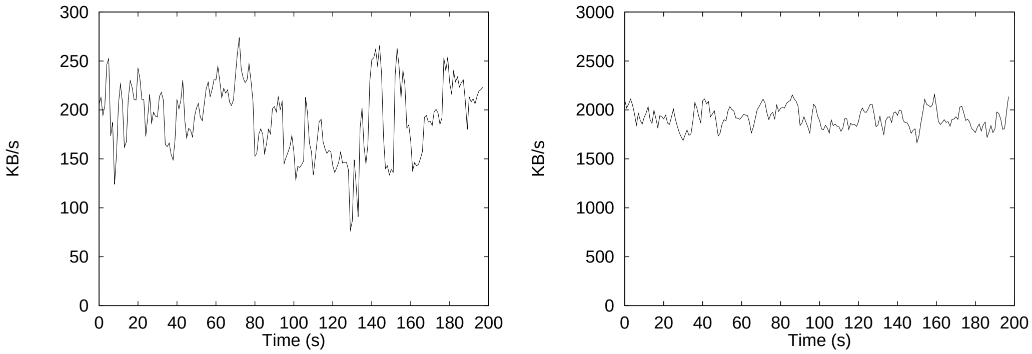


Figure 5: Variation in bit rates of video clients: (a) variation in the average bit rate of the Frasier MPEG-1 trace over one second intervals. (b) variation in the aggregate bit rate of 10 MPEG-1 clients.

2.3 Workload Generation

To compare the performance of the integrated server to the partitioned and ideal integrated servers (Figure 4), we need to evaluate their performance at different operating points (i.e., text and video loads). Since most available file system traces contain various parts of the operating region (these regions are not clearly identifiable in the trace), it becomes difficult to employ them for studying performance at any particular operating point. Hence, we chose to systematically explore the complete space by using a *synthetic load generator*.

Our synthetic workload generator selects an operating point by fixing the number of text and video clients accessing the file system. Let t ($0 \leq t \leq T$) and v ($0 \leq v \leq V$), respectively, denote the number of text and video clients. Each text client accesses a randomly selected file; the inter-arrival time between successive requests issued by each client is exponentially distributed, and the amount of data retrieved by each request is normally distributed. We varied both the mean inter-arrival time and the mean request size in our experiments; due to space constraints, we only present simulation results for a mean request size of 32KB and a mean inter-arrival time of 1s.

Each video client randomly selects a video file and then initiates retrieval from a random point in that file. Due to the periodicity and sequentiality of access, the video server services requests from the clients by proceeding in terms of periodic *rounds*, accessing f frames for each client during each round [20]. For the simulations, we use $f = 30$ and the round duration of 1 second. For variable bit rate (VBR) encoded video streams, the size of f frames may vary from one round to another (see Figure 5). The workload generator models this behavior by determining the amount of video data accessed during each round using VBR encoded MPEG-1 traces (see Table 2).

Table 2: Characteristics of MPEG-1 traces

MPEG File	Encoding Pattern	Length (frames)	Bit rate Mb/s
Frasier	$I(BBP)^3BB$	5960	1.49
Newscast	$I(BBP)^3BB$	9000	2.33
Flintstones	$I(BBP)^3BB$	9000	1.67
Olympics	$I(BBP)^3BB$	9000	1.49

Table 3: Characteristics of the Auspex NFS trace

Number of read/write operations	218724
Average bit rate (original)	218.64 KB/s
Average bit rate (with 64MB cache)	83.91 KB/s
Average inter-arrival (original)	9.14 ms
Average inter-arrival (with 64MB cache)	22.53 ms
Average request size	2048.22 bytes
Peak to average bit rate (1s intervals)	12.51

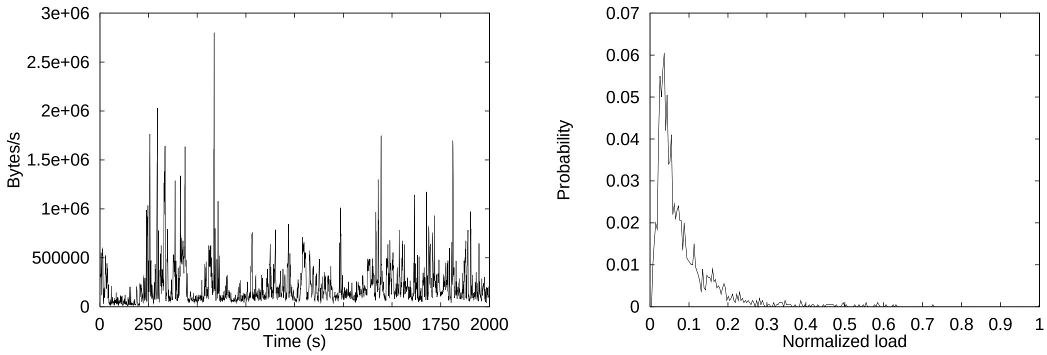
We compare the performance of the two architectures under bursty workloads in two steps. In the first step, we utilize bursty text load, and assume fixed video load. Specifically, to generate bursty text workload, we use a portion of the NFS traces gathered from an Auspex file server at Berkeley [5]; the characteristics of these traces are shown in Table 3. To derive the text workload, we first filter out requests that would be absorbed by a 64MB LRU buffer cache at the file server, and assume that the remaining requests result in disk accesses. Figure 6 illustrates the characteristics of the resulting text workload. For each experiment, the number of video clients accessing the server is kept fixed; each video client accesses a VBR encoded MPEG-1 video stream. We utilize traces to generate the workload from individual video clients.

In the second step, we assume that both video and text workloads are bursty. Since very few, if any, video servers have been deployed, traces that demonstrate the bursty nature of video access are not available. Hence, we hypothesize that the burstiness of video clients is similar to the text clients and then use textual traces to deduce the variability in video load. By using the distributions of text and video loads derived from traces (rather than using the trace itself), we can compute the expected performance of the two architectures. Thus, if $p(t,v)$ denotes the probability of the server being at operating point (t,v) (computed from traces) and $d(t,v)$ denotes the difference in the response times of the partitioned and the integrated architectures at a load (t,v), then the expected difference in the performance of the two architectures is $\sum_t \sum_v d(t,v) \cdot p(t,v)$.

3 Experimental Evaluation

The objective of this section is to answer two questions:

- Does the integrated architecture yield better performance in the shaded region in Figure 4? How does the capacity of an integrated server compare to that of an ideal integrated server depicted in Figure 4?



(a) (b)

Figure 6: Characteristics of the NFS traces: (a) average bit rate of NFS traces over one second intervals; this demonstrates the bursty nature of the trace. (b) the normalized load distribution function.

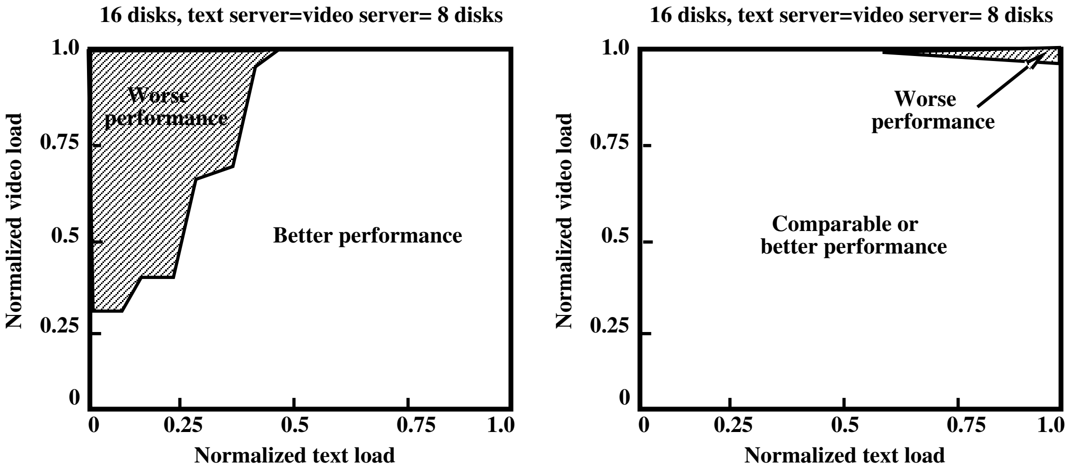
- What is the difference in performance of the integrated and partitioned architectures in the presence of bursty workloads?

We address the first question in Sections 3.1 and 3.2, and the second question in Section 3.3. Finally, we evaluate the sensitivity of our results to the particular choice of experimental parameters in Section 3.4.

3.1 Comparison of Partitioned and Integrated Systems

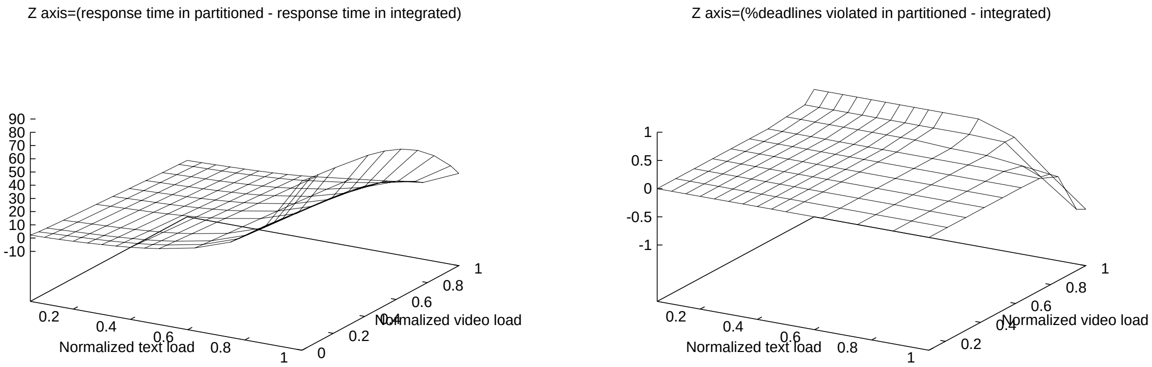
To compare the performance of the partitioned and the integrated systems in the shaded region in Figure 4, let us first define the region by determining the values of T and V, which, respectively, denote the maximum number of text and video clients supported by the partitioned server. To determine T, we increased the text workload until the average response time of a request yielded by the text server in the partitioned architecture exceeded a threshold τ_t. For $\tau_t = 100ms$, we obtained $T = 200$. To determine V, we increased the video workload until the percentage of request deadlines violations yielded by the video server in the partitioned architecture exceeded a threshold τ_v. For $\tau_v = 1\%$, we obtained $V = 102$.

Given the values of T and V, we varied the text and video loads in the ranges $[0, T]$ and $[0, V]$, respectively, in increments of 10; and for each combination (t, v), we measured the average response time of text requests and the percentage of deadlines violations for video requests in both partitioned an integrated servers. The results of these 231 experiments are summarized in Figures 7 and 8. The X and Y axes plot normalized values of text and video workloads; a normalized load of 1 corresponds to T text clients and V video clients. The non-shaded regions in Figure 7 indicate workloads at which the integrated server either yields comparable performance or outperforms the partitioned server; the shaded region identify regions where the partitioned system outperforms the integrated server. The figure demonstrates that sharing of disk bandwidth enables the integrated server to provide comparable or better performance over a large operating region. In what follows, we discuss in detail the results of our experiments; first for text clients and then for video clients.



(a) Response time (b) Percentage of deadlines violated

Figure 7: Performance under different workload mixes. The integrated server yields worse performance in the shaded areas.



(a) Response time (b) Percentage of deadlines violated

Figure 8: Performance under different workloads. The figure quantifies the difference in response times and percentage of deadlines violated in integrated and partitioned servers. The figure shows that the integrated server outperforms the partitioned server by a significant amount in many regions, while it under-performs its counterpart in other regions by only a small amount.

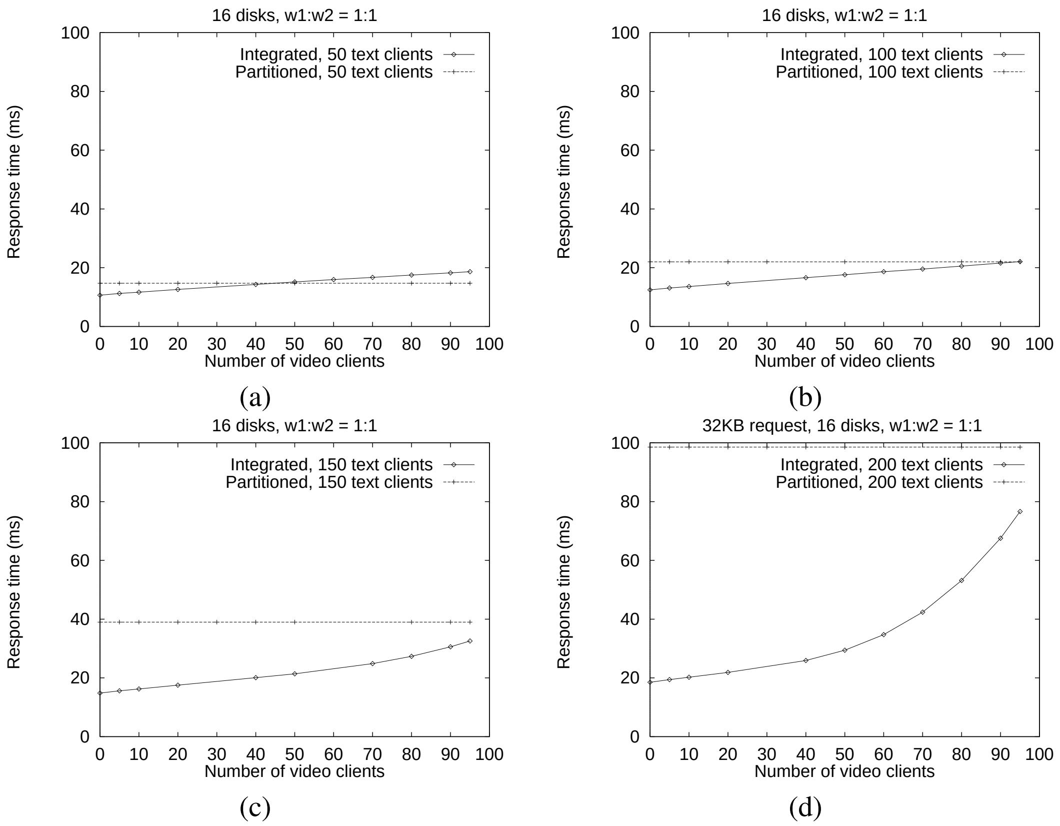


Figure 9: Response time of interactive text requests in partitioned and integrated servers. Figures (a) through (d) plot the variation in response times for different video workloads and a text load of 50, 100, 150 and 200 clients, respectively.

3.1.1 Performance of Text Clients

Figure 9 plots the variation in average response time of text clients for *different video* loads and a *fixed text* load. Since text and video requests access mutually exclusive set of disks, the response time of text requests in the partitioned server is independent of the video load. On the other hand, even though the integrated server uses the Cello disk scheduling algorithm to isolate text requests from video requests, the isolation is not total. Hence, the response time of text requests increases slowly with increase in video load. This increase can be attributed to two factors. First, increasing the video load increases the probability of a text request arriving when a video request is being serviced by the disk. Since requests in service cannot be preempted, the text request must wait until that request has been serviced. Second, increasing the video load also reduces the slack available to service text requests. Cello schedules a text requests prior to a video request only if sufficient slack is available; hence, reduction in slack yields an increase in the queuing delays and the response times observed by text requests.

Figure 10 compares the response time of text clients in the two architectures for *different text* loads and a *fixed video* load. As expected, increasing the text load causes the response time of text requests to increase in both servers, albeit the increase is larger in the partitioned server. To understand this behavior, consider the two factors that contribute to the response time of a request—service time and queuing delay. The service time of a request—defined as the summation of the seek time, rotational latency and transfer time incurred in servicing a request—in

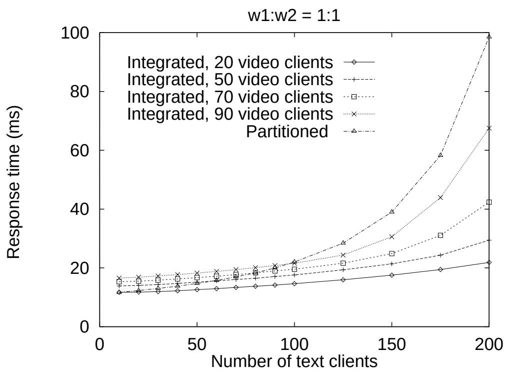
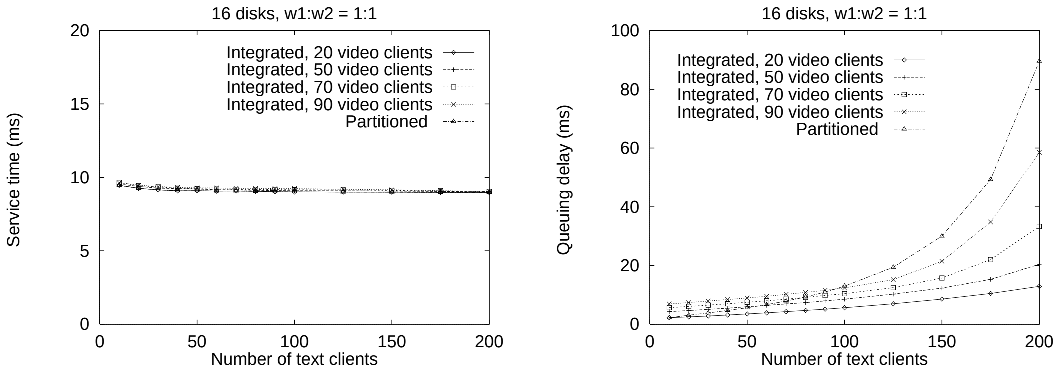


Figure 10: Response time of interactive text requests in partitioned and integrated servers. The figure plots the variation in response times for different text workloads.



(a) Service time (b) Queuing delay

Figure 11: Various components of the text response time. Figures (a) and (b) show the service time and the queuing delay incurred by a request for different workloads. The figures show that the service time of a request is largely independent of the workload, whereas the queuing delay depends on the load.

the average case depends on the physical characteristics of the disk and the amount of data being retrieved from disk. Hence, the service time of a request is *largely independent of the load* (see Figure 11(a)). In contrast, the queuing delay incurred by a request is *completely governed by the load* (see Figure 11(b)). Since text files are striped across a larger number of disks in the integrated architecture, the number of disks servicing text requests is larger than that in the partitioned system. This results in a smaller number of text requests per disk, and hence, shorter queues at each disk. Consequently, the integrated server yields better response times over a range of video loads. In fact, at heavy text loads, the queuing delay dominates the response time, causing the integrated server to outperform its counterpart *regardless of the video load* (see Figure 10 and 11(b)).

3.1.2 Performance of Video Clients

Figure 12 plots the percentage of deadlines violated for video requests for different values of text load with varying number of video clients. As shown in the figure, at light and moderate video loads, both servers meet deadlines of all real-time requests regardless of the text load. At heavy video load and light text load, the percentage of deadline

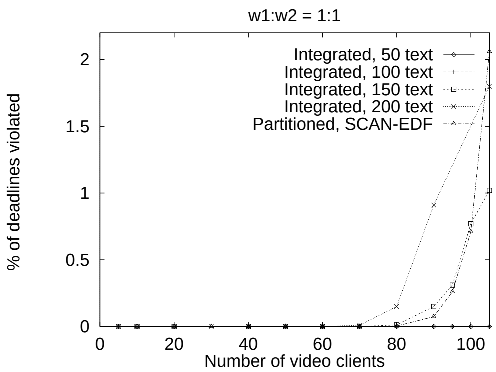


Figure 12: Percentage of deadlines violated for video requests in partitioned and integrated servers.

violations is comparable in both the servers. However, at heavy video and text load, the integrated server has slightly higher deadline violations than the partitioned server. This is due to two reasons:

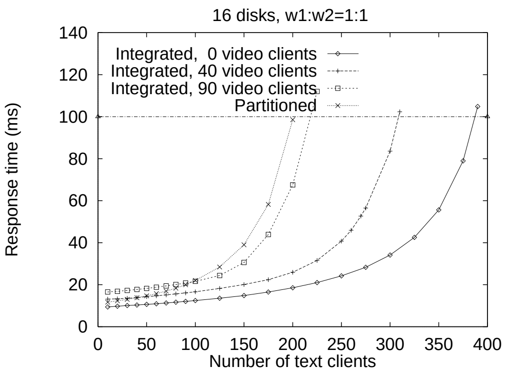
1. Switching between application classes causes the Cello disk scheduling algorithm employed by the integrated server to incur a higher seek and rotational latency overhead than the SCAN-EDF algorithm employed by the partitioned server.

2. It has been shown in [20] that, for a fixed number of video clients, increasing the number of disks in the server increases the load imbalance across the disks, leading to a higher percentage of deadline violations. Since the integrated server uses twice the number of disks to service video clients, it has a higher load imbalance resulting in a higher percentage of deadline violations. At light text loads, however, the effect of increased load imbalance is offset by the available unused bandwidth.

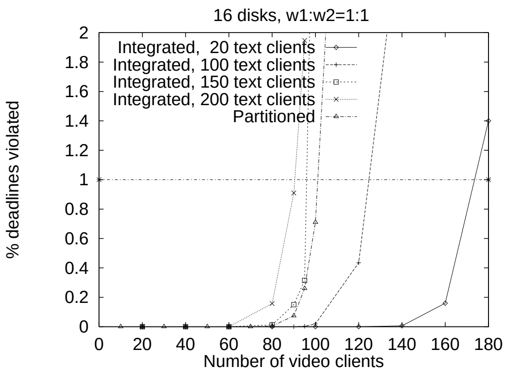
We conclude from our experiments thus far that in the shaded region (namely, $0 \leq t \leq T$ and $0 \leq v \leq V$), the integrated server yields higher performance improvement for text clients than video clients (see Figures 7 through 12). This is because, an integrated server utilizes any unused disk bandwidth allocated to video clients to reduce the average response time for text requests. For video requests, on the other hand, since the value of V is chosen such that the percentage of deadlines violations in the partitioned server does not exceed $t_v = 1\%$, there is little room for achieving significant reduction in the percentage of deadline violations. Hence, for the entire operating region, the difference in the percentage of deadline violations in the integrated and partitioned architecture is marginal. In the next section, we will demonstrate that both classes benefit from increased capacity yielded by the integrated architecture.

3.2 Capacity of an Integrated Server

As explained in Figure 4, an integrated server can operate in regions beyond the shaded area, whereas a partitioned server saturates at such workloads. In the previous section, we determined the performance of an integrated server in the shaded region and compared it to that of a partitioned server. In this section, we determine how the capacity of an integrated server compares with that of the ideal.



(a)



(b)

Figure 13: Capacity of the integrated server: (a) text capacity, and (b) video capacity

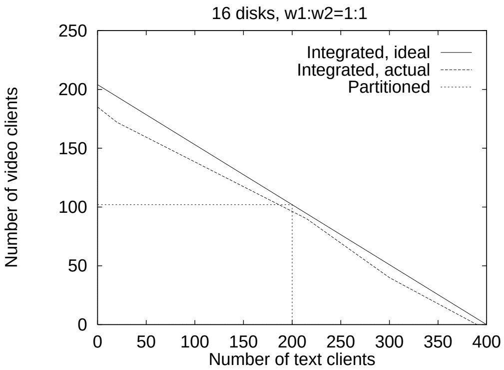


Figure 14: Capacity comparison of the partitioned, the ideal integrated, and the ideal integrated server architectures

An integrated server can support t text and v video clients simultaneously if the response time of text clients at that load is less than threshold $\tau_t = 100ms$ and the percentage of deadlines violated of video clients is less than threshold $\tau_v = 1\%$. We first determined the number of text clients that could be supported at various video loads. To do so, we varied the text load until the threshold τ_t was reached. Figure 13(a) plots the variation in response time for text clients for different video loads. Next, we determined the number of video clients that could be supported at various text loads. To do so, we increased the video load until the percentage of deadlines violated exceeded threshold τ_v. Figure 13(b) plots the percentage of request deadlines violated for different text load. Figure 14 combines the results of these experiments and plots the number of video and text clients that can be supported simultaneously by an integrated server.

Recall that, for $\tau_t = 100ms$ and $\tau_v = 1\%$, the partitioned server can support $T = 200$ text and $V = 102$ video clients. Since the integrated server, in our experiments, utilizes twice as many disks as the text and the video servers in the partitioned architecture, ideally, the integrated server should be able to support up to 400 text clients when there is no video load, and 204 video clients when there are no text clients. Furthermore, the performance of the ideal integrated server will scale linearly between these two extremes. Figure 14 compares the capacity of a partitioned, an ideal integrated, and an integrated servers. The figure shows that:

- The capacity curve of the integrated server is indeed similar to that of the the ideal integrated server; hence an integrated server can support approximately twice the number of clients from an application class when the other class does not use its fair share. However, there is a small degradation (about 5%) in capacity as compared to the ideal scenario.

- The partitioned server can support 200 text and 102 video clients simultaneously, whereas the integrated server can support 192 text and 97 video clients simultaneously. Thus, there is about 4.3% percentage degradation in the capacity of the integrated system as compared to the partitioned system.

The decrease in capacity of the integrated servers vis-a-vis partitioned and ideal integrated servers is due to the following reasons:

- *Effect of video clients on text requests:* The presence of video clients reduces the slack available to service text clients as well as increases the probability of a disk being busy when a text request arrives. Both factors require a text request to wait before being serviced. The resulting increase in response time causes the threshold τ_t to be reached at a smaller load, resulting in a reduction in capacity.

- *Increased load imbalance:* Since the integrated server uses $(D_1 + D_2)$ disks to service text and video clients, it has a higher load imbalance as compared to a partitioned server that uses D_1 and D_2 disks each to service text and video requests, respectively [20]. An increased load imbalance results in an increase in queuing delay and response time for text requests, which in turn reduces capacity. It also causes the most heavily loaded disk to reach saturation at a lighter video load. Since number of video clients supported by a server reaches its capacity when the most heavily loaded disk in the array reaches saturation (increasing the video load beyond this point results in an increasing number of request deadline violations), this results in a reduced video capacity [20].

- *Idiosyncrasies of the scheduling algorithm:* Since the Cello disk scheduling algorithm employed by the integrated server must switch between various classes, it incurs higher seek and rotational latency overheads than a partitioned system. This results is a reduction in throughput, and hence, capacity.

3.3 Performance for Bursty Workloads

In the previous two subsections, we compared: (1) the performance of the integrated and partitioned servers in the operating region defined by the ranges $[0, T]$ and $[0, V]$, and (2) the capacity of the integrated server with the ideal. For both scenarios, to determine the system behavior at different operating points, we used synthetic workloads for text clients that were non-bursty in nature. However, real-life workloads exhibit substantial burstiness at multiple time scales [5] and vary dynamically over time. Consequently, even when average load is substantially smaller than capacity, such workloads can contain periods of intense bursts causing transient overloads in the system. In this section, we compare the performance of the two architectures in the presence of such bursty workloads. We first examine the impact of burstiness along a single dimension (i.e., bursty text and fixed video loads) and then evaluate the performance when both text and video loads vary over time.

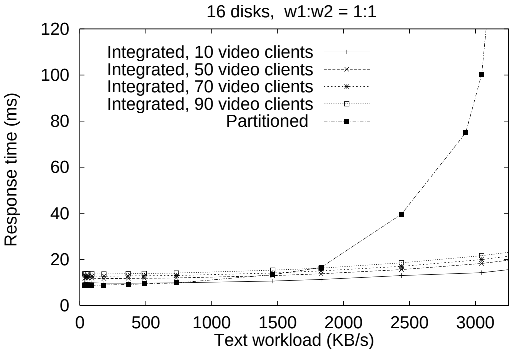


Figure 15: Performance using bursty text workloads.

3.3.1 Impact of Bursty Text Loads

To evaluate the performance of the two architecture in the presence of bursty text loads, we use the NFS trace described in Section 2.3. To generate different workloads using the same trace, we scale the timestamp associated with each request by a factor $\mathcal{S}$. Such scaling of timestamps changes the inter-arrival times between requests; $\mathcal{S} > 1$ increases the average inter-arrival time, while $\mathcal{S} < 1$ decreases it.

To determine the performance of text clients, we fix the video load and measure the response time assuming bursty text loads. Note that, the text loads used for our experiments were such that the *average* load was always smaller than capacity (i.e., was within the shaded region). Since the NFS trace has a peak to average ratio of 12.5 (see Table 3), the *peak* load causes the server to temporarily saturate during periods of intense bursts. Figure 15(a) compares the response times of the two servers. The figure shows that transient overloads cause the average response time yielded by the partitioned system to be much higher than the integrated server. This is because, as shown in Section 3.2, sharing disk bandwidth enables the integrated server to handle up to twice the number of clients from a particular application class if other class is not using up its share of the disk bandwidth. Consequently, the integrated server can adapt to changing load conditions and handle transient overloads that saturate the partitioned system. This results in average response times that are substantially smaller than that in the partitioned system. For instance, at a load of 3000KB/s, which corresponds to a utilization level of 50%, the response time is smaller by a factor of 6.

Next, we study whether a bursty text load causing transient overloads affects the performance of video clients in the integrated server. To do so, we computed the percentage of deadlines violated for video requests for various combinations of text and video loads. Our experiments showed that both servers are able to meet *all* request deadlines at low to moderate levels of utilization. A small number of requests deadlines were violated at very high levels of utilization in both servers (for instance, at 90 video clients and a text load of 2500KB/s, less than 50 requests deadlines out of 14000 requests were violated in both servers (i.e., 0.3% violation)). This demonstrates that Cello disk scheduling algorithm employed by the integrated server is able to isolate video requests from bursty text loads, and thereby provide performance that is comparable to the partitioned server.

3.3.2 Impact of Varying Text and Video Loads

Due to the unavailability of video workload traces, in Section 3.3.1 we used a synthetic workload generator that kept number of video clients fixed and used VBR MEG traces to generate a varying load from each client. In practice, the

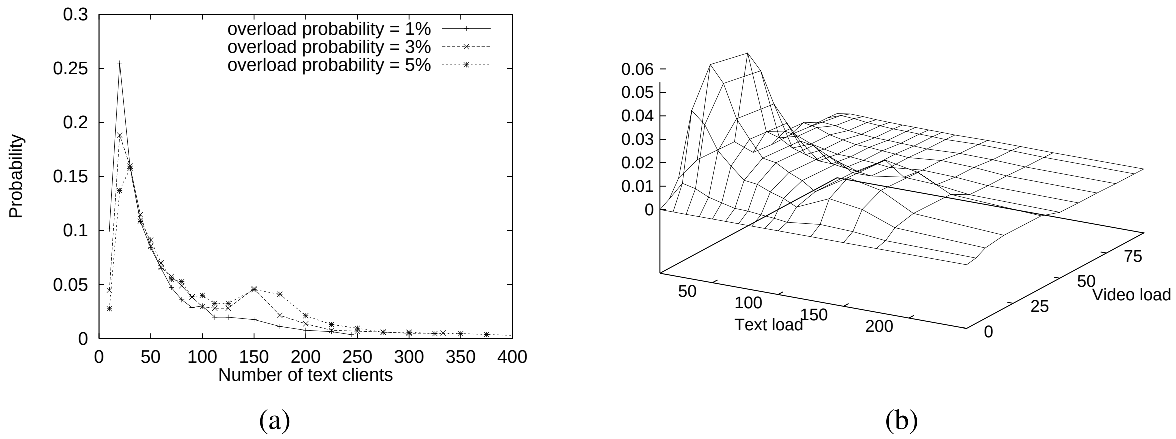


Figure 16: (a) Load distributions for text clients computed using different overload probabilities (b) The combined load distribution $p(t, v)$ for $\delta_1 = 0.01$.

video load (i.e., the number of clients accessing the server) also varies over time. Hence a natural question is: what is the expected performance of the two architectures in the presence of varying text and video loads? To answer this question, we hypothesize that the burstiness in the load distribution of video clients is similar to that of text clients and use the load distribution for text clients (derived from traces) to determine that for video clients.[1] Using these distributions, we can compute the probability $p(t, v)$ of t text and v video clients simultaneously accessing the server. Let $d(t, v)$ denote the difference in response times yielded by the two servers at the operating point (t, v). The expected improvement in response time of the integrated server is then

$$E(I) = \sum_t \sum_v d(t, v) \cdot p(t, v) \tag{1}$$

Thus, computing $E(I)$ requires us to first compute $d(t, v)$ and $p(t, v)$.

To determine $d(t, v)$, let $r_{part}(t)$ and $r_{int}(t, v)$, respectively, denote the response times for text requests yielded by the partitioned and integrated servers when the load consists of t text and v video clients. Then $d(t, v)$ is computed as

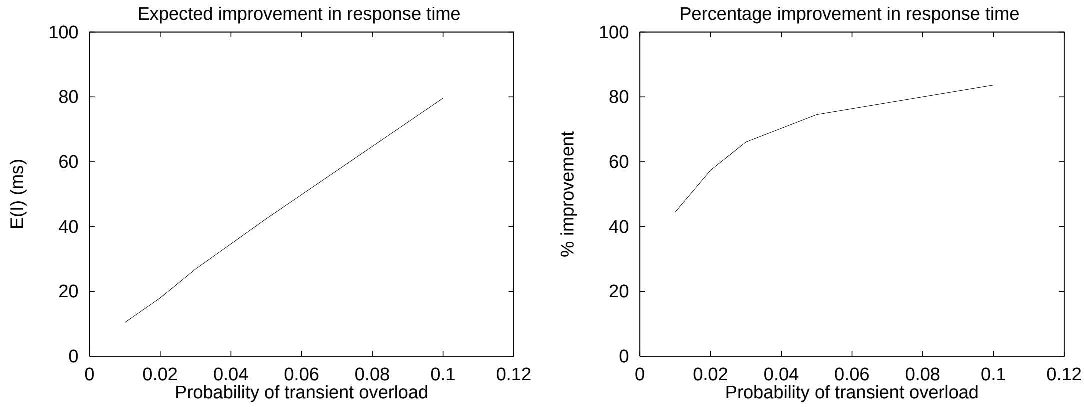
$$d(t, v) = r_{part}(t) - r_{int}(t, v) \tag{2}$$

Note that $d(t, v) > 0$ indicates that the integrated server yields better response times than the partitioned server and vice versa. Figure 8(a) plots $d(t, v)$ for different text and video loads.

The probability $p(t, v)$ of operating at a specific operating point is best determined from traces of real workloads. Due to the unavailability of traces from file systems that simultaneously support video and text applications, we make the following assumptions to derive $p(t, v)$ from textual workload traces.

- First, we assume that burstiness in load distributions of text and video clients are similar and hence, the *normalized* load distributions for video and text are identical. That is, $p(t)$ for a normalized text load of $[0, 1]$ is identical to $p(v)$ for a normalized video load of $[0, 1]$.

[1]Although time-scales at which text and video clients arrive are different (video loads change more slowly as compared to text loads), we are only interested in the peak to average variation of the load distribution here.



(a) (b)

Figure 17: Improvement in response time in the integrated server. Figures (a) and (b) show the expected improvement and the percentage improvement in response time for different probabilities of transient overload.

- Second, we model transient overloads by assuming that the probability that the text and video load exceeds capacity is δ_1 and δ_2, respectively. For text loads, we set δ_1 to a small value. Since video loads are constrained by admission control, we assume that our bursty video load never exceeds capacity (i.e., $\delta_2 = 0$). Then, the $[0, 1]$ ranges of the normalized load distributions $p(t)$ and $p(v)$ can be remapped to $[0, T']$ and $[0, V']$, where T' and V' denote the peak text and video loads such that $P(T < t \leq T') = \delta_1$ and $P(V < v \leq V') = \delta_2$.

- Third, we assume that the text and video loads accessing a server are independent of each other. Hence, $p(t, v) = p(t) \cdot p(v)$.

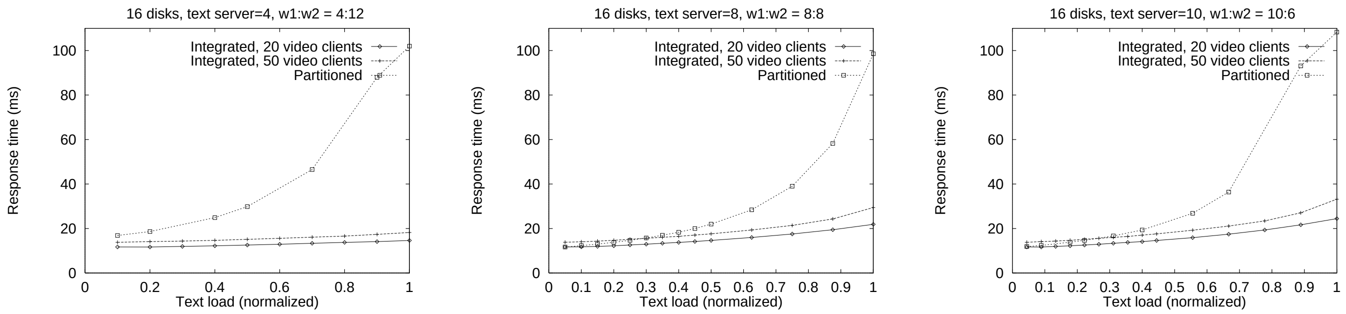
Thus, given $p(t)$ obtained from textual traces (see Figure 6(b)), we can compute $p(t, v)$.

To evaluate the effect of different peak loads, we varied δ_1 and constructed several distribution functions $p(t)$. Figure 16(a) shows $p(t)$ constructed using $\delta_1 = 0.01, 0.03$, and 0.05, respectively. The figure shows that the resulting distributions are heavy-tailed, where the average load is smaller than capacity while the peak load can be substantially larger than capacity (the distributions have average loads of 48.4, 64.7, and 76.9 clients, respectively and peak loads of 244, 333, and 401 clients). Figure 16(b) shows the $p(t, v)$ computed from one such distribution $p(t)$.

Figure 17 plots the expected improvement in response time $E(I)$ and the percentage improvement in response time for these distribution functions. The figure shows that the expected gain is always positive, and for the distributions considered the percentage improvement ranges from 40%–80%. Moreover, the gain increases with increasing average load since the difference in response times $d(t, v)$ is larger at higher loads (see Figure 8).

3.4 Effect of Partition Size

Our experiments so far have assumed that the disks in a partitioned system are equally partitioned among the text and video servers. To determine the sensitivity of our results to the partition size, we varied the partition size and computed the performance of the two architectures using synthetic workloads. Figure 18 plots the response time of text requests for various partition sizes (the bandwidth assigned to the two application classes in the integrated server



(a) text = 4 disks
video = 12 disks

(b) text = 8 disks
video = 8 disks

(c) text = 10 disks
video = 6 disks

Figure 18: Effect of varying the number of disks assigned to the text and video classes on the response time.

correspond to the fraction of the total number of disks assigned to the two servers in the partitioned system). The figure demonstrates that, even when the partition size is varied, the behavior of the response time graphs yielded by the two architectures remains similar. This implies that, regardless of the partition size, the expected response time of the integrated server continues to be smaller than that of its counterpart. Hence, the expected improvement $E(I)$ is always positive. We are in the process of conducting more extensive experiments for various partition sizes and will include these results in the final version of the paper.

4 Concluding Remarks

Integration—supporting multiple application classes with heterogeneous requirements—is an emerging trend in networks, file systems, and operating systems. In this paper, we evaluated two architectural alternatives—partitioned and integrated—for designing next generation file systems. Whereas a partitioned server employs a separate component file system for each application class, an integrated file server shares its resources across all application classes. We evaluated the performance gains achieved by the integrated architecture as a result of sharing disk bandwidth between application classes. We demonstrated that though the problem of sharing disk bandwidth is conceptually similar to that of sharing network link bandwidth in integrated services networks, the arguments that demonstrate the superiority of integrated services networks are not applicable to file systems. To experimentally evaluate the efficacy of sharing disk bandwidth, we considered two application classes, text and video, and for these application classes, showed that an integrated server: (i) yields *better performance* than its counterpart over a large operating region but has slightly worse performance in a small region, (ii) has a *larger capacity* since it can support a larger number of clients from a class when the other class does not use its fair share, and (iii) is *self-adapting* since larger capacity and sharing of disk bandwidth enable it to better handle bursty loads that cause transient overloads.

To derive these results, we first compared the performance of the two architectures when the load is less than the maximum that can be supported by the partitioned system (the shaded region in Figure 4). Our experiments demonstrated that the integrated server outperforms the partitioned server by a significant amount in a majority of this region and yields slightly worse performance than its counterpart in small region. Specifically, text requests see a worse response time under low text and heavy video loads (upper left hand corner of the region), whereas video request see a larger number of deadline violations under heavy text and video loads (upper right hand corner of the region). For video clients, we showed that, in spite of the potential interference from text requests, the integrated server is able to provide matching performance to video requests (by meeting a comparable number of request deadlines).

Next, we determined if the capacity of an integrated server scales up to $\frac{D_1+D_2}{D_2} * V$ video and $\frac{D_1+D_2}{D_1} * T$ text clients. We demonstrated that the capacity curve of an integrated server is indeed similar to that of the the ideal scenario; an integrated server can support a larger number of clients from an application class when the other class does not use its fair share. However, due to load imbalances in the system and the idiosyncrasies of the disk scheduling algorithm, there is a small (about 5%) degradation in the number of text and video clients supported as compared to the ideal server.

Finally, we demonstrated that the ability of an integrated server to support a larger number of clients enable it to adapt to bursty workloads that can cause transient overloads. Specifically, we showed that, when the server is operating at 50% utilization, in the presence of bursty text loads, the average response time yielded by the integrated server is smaller by a factor of 6 as compared to the partitioned server. On an average, the integrated server yields a 40%–80% improvement in response time over its counterpart. Moreover, such bursty loads have little or no effect on the performance of video clients.

In this paper, we evaluated the two architectures with respect to their performance. However, factors other than resource utilization and performance may also influence the choice of a particular architecture. For instance, partitioned file systems are easy to design and implement, since techniques for designing file systems optimized for a single application class are well understood. Furthermore, the presence of legacy file systems or specialized applications that need custom hardware and software may dictate the use of partitioned or even separate servers. In contrast, system administration costs may favor the use of integrated servers over partitioned or multiple disparate servers. Although the need for supporting multiple application classes within an integrated server increases file system complexity and development costs, such file systems generally facilitate easy integration of new application classes; allowing the development cost to be amortized over time as new application classes are added. Adding a new application class in the partitioned system, on the other hand, requires the development of a new component file system. Thus, in summary, the choice between the partitioned and integrated architectures is dependent on the needs of a particular environment, which govern the balance between these tradeoffs.

References

[1] P. Barham. A Fresh Approach to File System Quality of Service. In *Proceedings of NOSSDAV'97, St. Louis, Missouri*, pages 119–128, May 1997.

[2] J.C.R. Bennett and H. Zhang. Hierarchical Packet Fair Queuing Algorithms. In *Proceedings of SIGCOMM'96*, pages 143–156, August 1996.

[3] E G. Coffman and M. Hofri. On the Expected Performance of Scanning Disks. *SIAM Journal of Computing*, 10(1):60–70, February 1982.

[4] E G. Coffman, L A. Klimko, and B. Ryan. Analysis of Scanning Policies for Reducing Disk Seek Times. *SIAM Journal of Computing*, 1(3):269–279, September 1972.

[5] M. Dahlin, C. Mather, R. Wang, T. Anderson, and D. Patterson. A Quantitative Analysis of Cache Policies for Scalable Network File Systems. In *Proceedings of ACM SIGMETRICS'94*, May 1994.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of ACM SIGCOMM*, pages 1–12, September 1989.

[7] P J. Denning. Effects of Scheduling on File Memory Operations. In *Proceedings of AFIPS SJCC*, pages 9–21, 1967.

[8] R. Geist and S. Daniel. A Continuum of Disk Scheduling Algorithms. *ACM Transactions on Computer Systems*, 5(1):77–92, February 1987.

[9] L. Golubchik, J. C. S. Lui, and R. R. Muntz. Reducing I/O Demand in Video-On-Demand Storage Servers. In *Proceedings of SIGMETRICS '95, Ottawa, Canada*, May 1995.

[10] M. Hofri. Disk Scheduling: FCFS vs. SSTF Revisited. *Communications of the ACM*, 23(11):645–653, November 1980.

[11] 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.

[12] C. Martin, P. S. Narayan, B. Ozden, R. Rastogi, and A. Silberschatz. The Fellini Multimedia Storage Server. *Multimedia Information Storage and Management , Editor S. M. Chung, Kluwer Academic Publishers*, 1996.

[13] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A Fast File System for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.

[14] G. Nerjes, P. Muth, M. Paterakis, Y. Romboyannakis, P. Triantafillou, and G. Weikum. Scheduling Strategies for Mixed Workloads in Multimedia Information Servers. In *Proceedings of the 8th International Workshop on Research Issues in Data Engineering (RIDE'98), Orlando, Florida*, February 1998.

[15] A.L. Narasimha Reddy and J. Wyllie. Disk Scheduling in Multimedia I/O System. In *Proceedings of ACM Multimedia'93, Anaheim, CA*, pages 225–234, August 1993.

[16] Timothy Roscoe. *The Structure of a Multi-Service Operating System*. PhD thesis, University of Cambridge Computer Laboratory, April 1995. Available as Technical Report No. 376.

[17] S. Shenker. Fundamental Design Issues for the Future Internet. *IEEE Journal of Selected Areas in Communications*, 13:1176–1188, September 1995.

[18] 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.

[19] P J. Shenoy, P. Goyal, S S. Rao, and H M. Vin. Symphony: An Integrated Multimedia File System. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking (MMCN'98), San Jose, CA*, pages 124–138, January 1998.

[20] P J. Shenoy and H M. Vin. Efficient Striping Techniques for Multimedia File Servers. In *Proceedings of the Seventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'97), St. Loius, MO*, pages 25–36, May 1997.

[21] T. Teorey and T. B. Pinkerton. A Comparative Analysis of Disk Scheduling Policies. *Communications of the ACM*, 15(3):177–184, March 1972.

[22] F A. Tobagi, J Pang, R Baird, and M Gang. Streaming RAID – A Disk Array Management System For Video Files. In *Proceedings of ACM Multimedia '93, Anaheim, CA*, pages 393–400, 1993.

[23] A. K. Tsiolis and M. Vernon. Group Guaranteed Channel Capacity in Multimedia Storage Servers. In *Proc. ACM Sigmetrics '97, Seattle*, pages 285–297, June 1997.

[24] R. Wijayaratne and A. L. N. Reddy. Providing QoS Guarantees for Disk I/O. Technical Report TAMU-ECE97-02, Department of Electrical Engineering, Texas A&M University, 1997.

[25] N. C. Wilhelm. An Anomaly in Disk Scheduling: A Comparison of FCFS and SSTF Seek Scheduling using and Empirical Model for Disk Access. *Communications of the ACM*, 19(1):13–17, January 1976.

[26] B L. Worthington, G R. Ganger, and Y N. Patt. Scheduling Algorithms for Modern Disk Drives. In *Proceedings of ACM SIGMETRICS'94*, pages 241–251, May 1994.