An Architectural study of Cluster-Based Multi-Tier Data-Centers

K. VAIDYANATHAN, P. BALAJI, J. WU, H. -W. JIN, D. K. PANDA

Technical Report
OSU-CISRC-5/04-TR25

An Architectural study of Cluster-Based Multi-Tier Data-Centers*

K. Vaidyanathan P. Balaji J. Wu H. -W. Jin D. K. Panda
Computer Science and Engineering
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210
{vaidyana, balaji, wuj, jinhy, panda}@cis.ohio-state.edu

Abstract

The phenomenal growth and popularity of cluster-based multi-tier data-centers has not been accompanied by a system-wide understanding of the various resources and their deployment strategies. Typical data-center workloads have a wide range of characteristics. They vary from high to low temporal locality (Zipf coefficient), large documents to small documents (download sites vs book stores), the number of documents and several others. However, the implications of the various system resources such as CPU, file system, disk, network, I/O on these different kinds of workloads have not been previously studied. In other words, the architectural characterization of cluster-based multi-tier data-centers has been incomplete in many respects.

In this paper we analyze several system level micro-benchmarks, based on the various resources mentioned above to identify the characteristics of different kinds of workloads. Depending on these characteristics associated with the workloads, making enhancements to some of the system resources becomes more critical with respect to the end performance compared to the others. We back up the observations of these workload characteristics with experimental results showing the impact of varying the above mentioned system resources on the particular workloads. We conclude the study by providing several solutions (e.g., file distribution strategies, memory file system for popular documents, etc.) that allow the design of efficient next generation cluster-based multi-tier data-centers in a manner more tightly coupled with the expected workload characteristics.

Keywords: Clusters, Multi-Tier Data-Center, TCP/IP, PVFS, Architectural Characterization

*This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #EIA-9986052, #CCR-0204429, and #CCR-0311542

1 Introduction

Cluster systems have become the main system architecture for a number of environments. In the past, they had replaced mainstream supercomputers as a cost-effective alternative in a number of scientific domains. On the other hand, with the increasing adoption of Internet as the primary means of interaction and communication, highly scalable and available web servers have become a critical requirement. Based on these two trends, several researchers have proposed the feasibility and potential of cluster-based multi-tier data-centers [25, 4]

A cluster-based multi-tier data-center is an Internet server oriented cluster architecture, which is distinguished from high performance computing systems in several aspects. In particular, data-centers are required to execute various server software that demand different system resources and characteristics; these are distinguished from high performance computing systems by using duplicated programs performing symmetric jobs to maximize parallelism.

Multi-tier data-centers generally consist of three tiers; the proxy tier, the application/web tier and the database tier. Accordingly, since each tier of the multi-tier data-center has different requirements and behavior, it is a challenge to analyze the various architectural components, such as file system, I/O, network protocol, etc., and their influence on each tier. Further, each kind of workload has its own characteristics, making one component more important than the other with respect to the end performance perceived by the clients. For example, small static file requests tend to be neither compute nor I/O intensive. However, their performance in most cases is highly sensitive to the caching ability of the file system due to their high access frequencies. Similarly, as we will see in the later sections, small to moderate file requests tend to have portions of compute and I/O interleaved. This causes an increase in the parallelism by using multiple CPUs to have a greater impact than increasing the CPU speed as such.

In this paper, we analyze several architectural compo-

nents of a cluster-based multi-tier data-center and identify their impact on the criticality of the performance of different workloads. In particular, we examine the impact of workload characteristics on the performance of the file system, I/O subsystem, network protocol, and compute system. Since typical web workloads have a wide range of characteristics, we consider various workload factors such as locality, size, and popularity. For each component, we analyze its performance on different workloads in the data-center environment. We believe that every workload has a critical characteristic which demands the performance of one or more architectural components in the cluster-based data-center environment. We try to identify these critical characteristics in the workload and provide analysis of various design alternatives to maximize the performance contribution of each component. Our study reveals several insights into the characteristics and the applicability of various emerging technologies in the data-center environment. For example, for moderate sized workloads, a manual distribution of files among SCSI disks based on popularity is found to provide comparable performance with the software RAID based disk system using SCSI disks.

To the best of our knowledge, this is a unique study that analyzes the various components of a cluster-based multi-tier data-center and their impact on the various workloads.

The remaining part of the paper is organized as follows: Section 2 provides a brief background about multi-tier data-centers, the Parallel Virtual File System (PVFS) and TCP/IP. In Section 3 we mention the workload and testbed that we used. In Section 4, we evaluate various file systems in the data-center environment in more detail. Section 5 deals with the evaluation of various disk systems. In Section 6 we discuss the impact of TCP/IP and the potential of checksum offloading benefits in the data-center environment. Section 7 provides some insights into the relevance of compute resources in data-centers. We conclude the paper in Section 8.

2 Background

In this section, we give a brief overview of the architecture of multi-tier data-centers, *PVFS*, the traditional host-based TCP/IP implementation.

2.1 Multi-Tier Data-Centers

Figure 1 represents a typical multi-tier data-center. The first tier in a multi-tier data-center consists of a cluster of nodes known as the edge nodes. These nodes can be thought of as switches (Layer 7) providing load balancing, security, caching, etc. The next tier is usually the front-end servers which are commonly known as proxy servers that provide web, messaging and various other services to clients. In traditional data-centers edge services are provided on the front-end tiers, commonly referred to as proxy servers. In this paper, we consider this kind of a traditional data-center. The next tier (mid-tier) contains the web-servers and application servers. These nodes apart from serving static or time invariant content, can fetch dynamic or time variant data from other sources and return the data to the end-user in a presentable form. The last tier of the multi-tier data-center is the database tier (back-end applications). It is used to store persistent data. This tier is compute or I/O intensive depending on the workload.

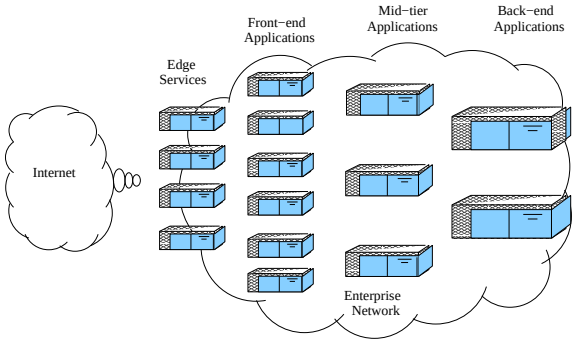


Figure 1. A Typical Multi-Tier Data-Center (Courtesy CSP Architecture design [28])

A request from a client is received by the edge/proxy servers. If this request can be serviced from cache, it is serviced. Otherwise, it is forwarded to the web/application servers. Static requests are serviced by the web servers by just returning the requested file to the client via the proxy server. This content may be cached at the proxy server so that subsequent requests to the same static content may be served from the cache. The application tier nodes handle dynamic content. The type of applications this tier includes range from mail servers to directory services to ERP software. Any request that needs a value to be computed, searched, analyzed or stored uses this tier. The back-end database servers are responsible for storing data persistently and responding to queries. These nodes are connected to persistent storage systems. Queries to the database systems can be anything ranging from a simple seek of required data to performing joins, aggregation and select operations on the data.

2.2 Parallel Virtual File System (PVFS)

PVFS [7] is a parallel cluster-based file system. It was designed to meet the increasing I/O demands of parallel applications. Figure 2 demonstrates a typical *PVFS* environment. As shown in the figure, a number of nodes in the cluster system can be configured as I/O servers and one of them (either

an I/O server or an a different node) as a metadata manager.

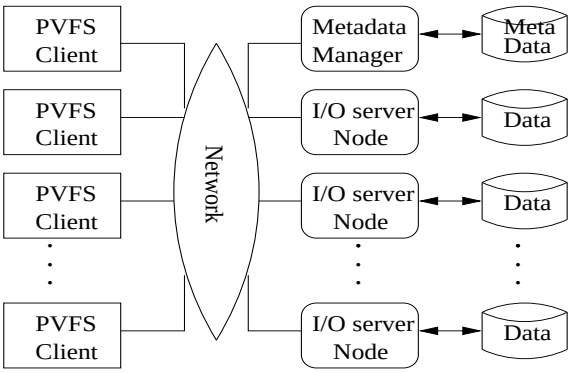


Figure 2. A Typical PVFS Setup

PVFS achieves high performance by striping a file across a set of I/O server nodes allowing parallel access to the file. It uses the native file system on the I/O servers to store individual file stripes. An I/O daemon runs on each I/O node and services requests from the client nodes. It supports both read and write requests.

A manager daemon running on a metadata manager node handles metadata operations involving file permissions, truncation, file stripe characteristics, etc., but does not participate in read/write operations. The metadata manager provides a cluster-wide consistent name space to applications.

PVFS supports a set of feature-rich interfaces, including support for both contiguous and non-contiguous accesses to memory and files [9]. PVFS can be used with multiple APIs: a native PVFS API, the UNIX/POSIX API, MPI-IO [33], and an array I/O interface called the Multi-Dimensional Block Interface (MDBI). The presence of multiple popular interfaces contributes to the wide success of PVFS in the industry. In this paper, we have studied *PVFS* in detail and proposed some enhancements to improve its performance contribution in the data-center environment.

2.3 TCP/IP Protocol Stack

Current day multi-tier data-centers has been using the traditional host-based TCP/IP [32, 34] stack for packet processing. TCP/IP deals with issues such as dividing the data passed to it from the application into appropriate sized chunks for the network layer below (segmentation), acknowledging received packets (reliability), setting timeouts to make certain the other end acknowledges packets that are sent, checking for possible data corruption (data integrity) and several others.

Traditionally, TCP/IP has not been able to take advantage of the high performance provided by the physical networks mainly due to the multiple copies and kernel context switches present in its critical message-passing path. A number of approaches [20, 24, 8, 29, 18, 10, 13, 17, 22, 15, 31, 16, 19, 12, 27, 35] have been devised to improve the performance given by TCP/IP. However, these approaches have had only limited success.

End systems incur CPU overhead for processing each network packet or frame. These per packet costs include the overhead to execute the TCP/IP protocol code, allocate and release memory buffers, and field device interrupts for packet arrival and transmit completion. TCP/IP implementations incur additional costs for each byte of data sent or received. These include overheads to move data within the end system and to compute and verify checksums.

Jacobson et. al., had proposed an optimization to integrate checksum and copy on the transmission side to reduce the per-byte overhead [11]. This optimization is used in the current 2.4 linux kernels for the sender side. On the receiver side, after receiving the packet to the kernel buffer, the TCP/IP stack verifies the data integrity by performing checksum computation and then copies the data into the user buffer.

A number of approaches have been proposed to minimize the per-packet and the per-byte overheads in the TCP/IP protocol implementation including jumbo frames and interrupt coalescing, which have been previously studied. In this paper, we focus on the TCP checksum offloading. The checksum offloading approach deals with offloading the data checksum computation and verification of TCP and IP to hardware.

3 Workload and Experimental Setup

In this section we describe the characteristics of the workload that we used in the paper.

3.1 Workload Characteristics

Different workloads have different characteristics. Some workloads may vary from high to low temporal locality, following a Zipf-like distribution [6]. Similarly workloads vary from small documents (e.g., online book stores, browsing sites, etc.) to large documents (e.g., download sites, etc.). Further, workloads might contain requests for simple cacheable static or time invariant content or more complex dynamic or time variant content via CGI, PHP, and Java servlets with a back-end database. Due to the varying characteristics of workloads, we classified the workloads in four broad categories:

- Static content (average file) based workload
- Dynamic content workload
- Zipf-like workload with varying Zipf factor

- Real workloads - world-cup trace

The static content based workloads generate requests for files which are on an average of a particular file size. Typically, browsing sites have files which are a few kilobytes in size. Streaming and download servers on the other hand, may have large audio and video files which are a few MBytes in size. In order to address all these workloads, we used has a wide range of file sizes from 1 KB to 64 MB. Data-centers also serve dynamic content which typically require some amount of computation before they can return the document to the user.

It has been well acknowledged in the community that most of the workloads follow a Zipf-like distribution [6]. Zipf law states that the relative probability of a request for the i'th most popular document is proportional to $1/i^{\alpha}$, where α is a factor that determines the randomness of file accesses. In our experiments, we have used two different kinds of workloads: (i) with a constant alpha value and varying working set size, (ii) with varying alpha values. For synthetic workloads, we varied the file sizes from 1 KB to 4 MB and for large workloads we varied file sizes till 64 MB. To validate our research findings, we also evaluated the system under real workloads like the world-cup trace [2].

3.2 Testbed Configuration

For all our experiments we used 3 clusters whose descriptions are as follows:

Cluster1: A cluster system consisting of 8 nodes built around SuperMicro SUPER P4DL6 motherboards and GC chipsets which include 64-bit 133 MHz PCI-X interfaces. Each node has two Intel Xeon 2.4 GHz processors with a 512 kB L2 cache and a 400 MHz front side bus and 512 MB of main memory. We used the RedHat 9.0 Linux distribution with the kernel.org SMP kernel version 2.4.22smp.

Cluster2: A cluster system consisting of 16 nodes which include 64-bit 33 MHz PCI interfaces. Each node has two Pentium III 1 GHz processors with a 256 kB L2 cache and a 400 MHz front side bus. We used the RedHat 7.1 Linux distribution with the kernel.org SMP kernel version 2.4.18.

Cluster3: A cluster system consisting of 16 nodes which include 64-bit 66 MHz PCI interfaces. Each node has four Intel Pentium-III 700 MHz processors with 1 MB L2 cache and a 400 MHz front side bus and 1 GB of main memory. We used the RedHat 7.1 Linux distribution with the kernel.org SMP kernel version 2.4.18.

All nodes in the three clusters were together connected using the Myrinet network with 133 MHz LANai 9.1 processors, connected through a Myrinet 2000 network. The GM version used is 1.6.3.

In all our experiments, we used a total of 640,000 requests with 8 clients on Cluster2 running 8 threads each. For all experiments, the data-center was configured with 4 proxies, 3 web/application servers and one database server. Depending on the experiment, we chose the cluster (1, 2 or 3) to configure the data-center on. The actual cluster used for each experiment is specified with the experiment descriptions in the respective sections. The Apache version used is 2.0.48.

For the I/O experiments we had the following setup with three disk systems (*IDE*, *SCSI*, *software RAID*). In the *IDE* case, a 40 GB Seagate ATA 100 disk (Model: ST340016A) is used. In the *SCSI* disk, a 18 GB FUJITSU (Model: MAM3184MC) with 1500rpm *SCSI* disk is used. In the *RAID* case, a RAID-0 software *RAID* is built on top of three SCSI disks. The *RAID* stripe size is 4 KB. In all cases, an *ext3fs* is mounted on the corresponding disk systems.

4 File System Requirements

In order to study the impact of the performance of the various file systems in a cluster-based multi-tier data-center, we performed various system-level micro-benchmarks and application level tests. We broadly considered both local file systems (*ext3fs* and *ramfs*) as well as cluster file systems (*PVFS*). In this section, we first analyze the peak performance achieved by these file systems and then try to study their impact on the client's response time in a multi-tier data-center environment. All experiments had been performed using *IDE* disks (configuration mentioned in Section 3.2).

4.1 Basic Performance of Different File Systems

As mentioned earlier, we considered three file systems: *ramfs*, *ext3fs* and *PVFS*. The ext3fs was mounted on the *IDE* disk mentioned in Section 3.2. In the *PVFS* setup, three I/O nodes are configured. The local file system on each I/O node is *ext3fs* on the *IDE* disk and the stripe size was 64KB.

File systems	Read (MB/s)	Write (MB/s)
ramfs	775	800
ext3fs	39	40
PVFS	136	110

Table 1. Basic Performance of Different File Systems

Table 1 shows the sequential read and write bandwidths of these three file systems. In *PVFS*, since data is striped onto three disks, each on a different node, when the access size is large enough to access the three I/O nodes together, we can realize almost three times the local *ext3fs* performance.

Figure 3a shows the peak read bandwidth achieved by all three file system for various read and write buffer sizes.

As mentione earlier, *PVFS* realizes thrice the bandwidth when the read buffer size is more than three times the stripe size. The *ext3fs* and *ramfs* realize a read bandwidth of 39 MBytes/sec and 775 MBytes/sec respectively. Figure 3b shows the peak write bandwidth achieved by *ext3fs*, *ramfs* and *pvfs*. *ext3fs* achieves a peak write bandwidth of 40 MBytes/sec compared to 110 MBytes/sec achieved by *PVFS* and 91 MBytes/sec achieved by *ramfs*. As mentioned above, the bandwidth of *ramfs* is very high since the data is already in memory. As read buffer size increases we realize three times the local disk read performance.

In the multi-tier data-center environment, web servers and proxy servers often use `sendfile(2)` system call to service data access requests. This is because `sendfile()` can combine a file read and a network write efficiently. However, when we conducted our experiments with Apache, we found that *PVFS* gives unstable results when the sendfile option is enabled in Apache. Therefore, in our tests, we disabled the sendfile option in Apache, i.e., the Apache server uses a file read and a network send to serve each request.

4.2 *PVFS* performance in a Data-Center Environment

In Figure 4a, we see the performance comparison of *ext3fs* and *pvfs* under two scenarios using (i) read/write calls. (ii) *sendfile*. *sendfile* is a useful user-level call which copies data between one file descriptor to another. The copy is done within the kernel and hence *sendfile* need not spend time transferrring data to and from user space. In our microbenchmark test, we realized that *sendfile* was giving unstable results. In Figure 4b, we see that *pvfs* read/write performs the best. *pvfs* takes advantage of the three disks and realizes an aggregate bandwidth of three disks whereas *ext3fs* realizes only a single disk bandwidth. Similarly we see that in Figure 5a *read/write* performs better as we increase the read buffer size from 64K to 192K. The default stripe size used in this experiment was 64 KB.

4.3 Impact of File System in Data-Center

As shown in Table 1, the three studied file system setups provide different raw performances. To show the impact of their performance on different data-center workloads, we conduct web benchmarks with five different workloads (Table 2).

Figure 6 (a) shows the throughput achieved on *ext3fs*, *ramfs* and *pvfs* for *class 0*, *class 1* and *class 2* workloads. With these workloads, since nearly all the files can be cached in the file system cache, *ext3fs* and ramfs do better. *PVFS* does worse due to two reasons: (i) there is no cache in the *PVFS* client side, therefore the Apache server needs to read data from the PVFS I/O nodes every time. Though data may be cached on the I/O node side, the performance is limited by the network performance; (ii) *open* and *close* operations in *PVFS* are much more expensive than both *ext3fs* and *ramfs* due to the distributed nature of *PVFS* (or many other cluster file systems[1]). In these workloads, accesses to small files are dominant causing the overheads of open and close operations to be significant compared to the time of reading data.

Class	File Sizes	Zipf alpha	Working Set Size
Class 0	1K - 250K	0.9	25 MB
Class 1	1K - 1MB	0.9	100 MB
Class 2	1K - 4MB	0.9	450 MB
Class 3	1K - 16MB	0.9	2 GB
Class 4	1K - 64MB	0.9	6 GB

Table 2. Workload Classification

Figure 6 (b) shows the throughput achieved on *ext3fs*, *ramfs* and *pvfs* for *class 3* and *class 4* workloads. In these workloads, the working set size is much larger than the file system cache. In addition, the number of accesses to the large files is also significant. Since *ramfs* cannot hold all the data, only results on *ext3fs* and *pvfs* are presented. In the class 3 workload, since most of files can still be cached in the file system cache in *ext3fs*, *ext3fs* performs better than *PVFS*. But the difference is reduced compared to the results in Figure 6 (a). In the *class 4* workload, the Apache server encounters very few cache hits. There are a significant amount of disk I/O accesses in the *ext3fs* case, including many accesses to large files. *PVFS*, on the other hand, provides higher raw performance to access large files. This benefit can be realized by applications in workloads such as the *class 4* workload.

4.4 Hybrid File System

As shown in Figure 6, caching has a significant impact on the performance of the Apache server. Ideally, we expect that the "hot" data files to always be cached. However, in a busy web server, particularly when an ISP hosts multiple websites, the behavior of the file system cache becomes unpredictable. It is highly possible that a large file which is seldom accessed may push many of the small but "hot" files out of the cache. Due to the high frequency of accesses of these small files, they tend to be highly sensitive to the caching capability of the file system. Thus, it is desirable

[1] Some Cluster file systems such as *NFS* optimize on file open and close operations by providing this control to the client itself when only one client is accessing the data. However, since in a typical data-center environment multiple nodes try to contact the file system server(s), such an optimization might not be possible.

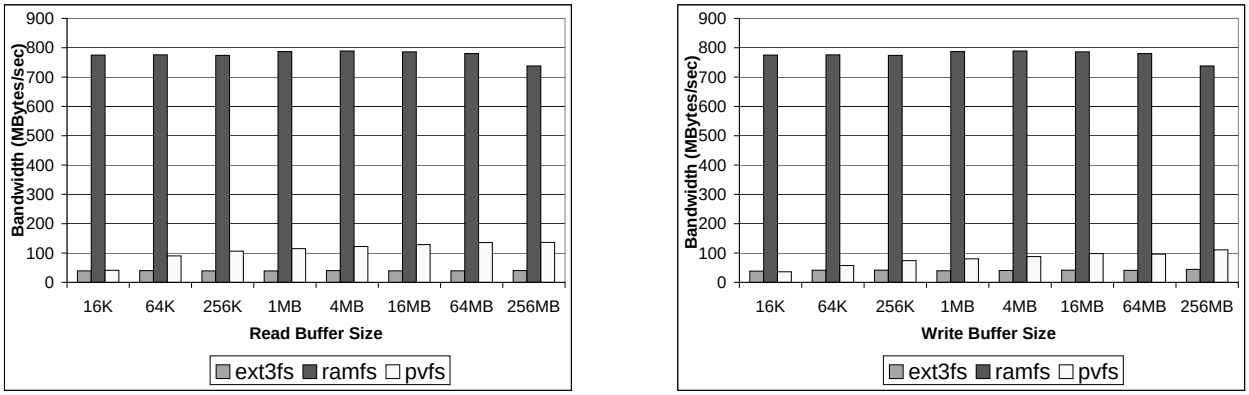


Figure 3. Performance of ext3fs, ramfs, pvfs (a) Read Bandwidth (b) Write Bandwidth

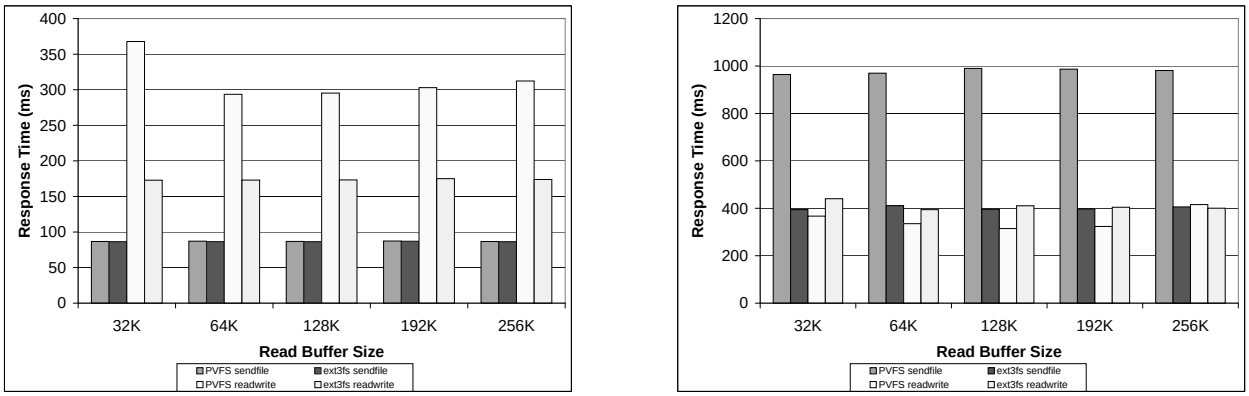


Figure 4. PVFS Performance for a 16 MB file (a) File in Cache (b) File not in Cache

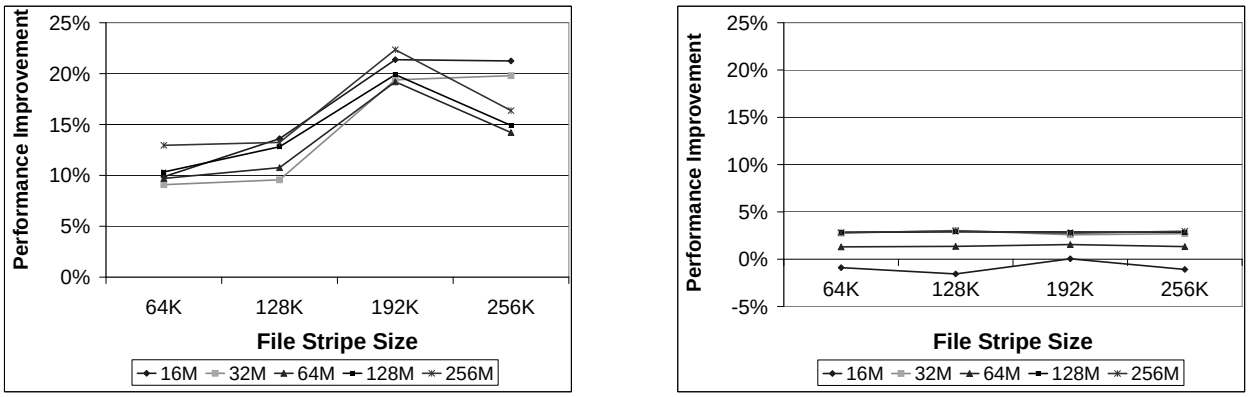


Figure 5. PVFS Performance for a 16 MB file with different read buffer sizes (a) *read/write* **(b)** *sendfile*

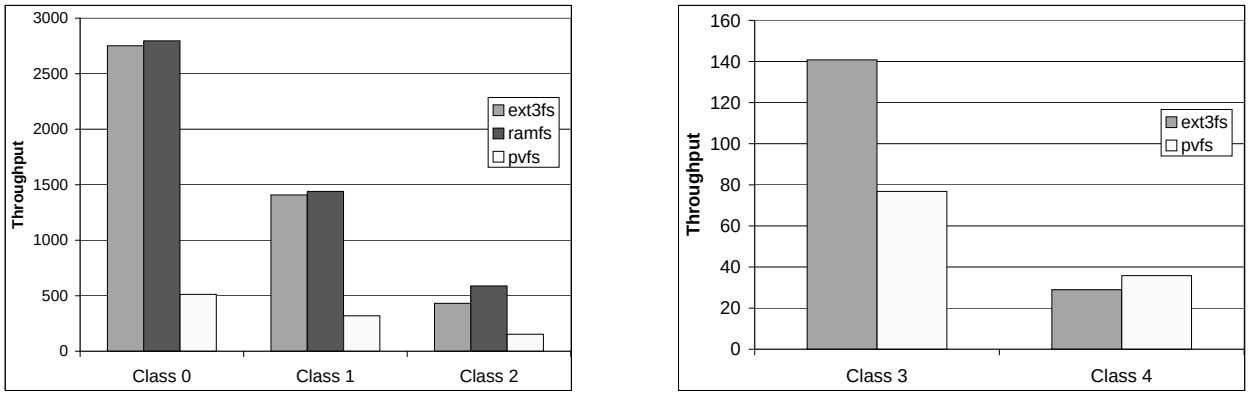


Figure 6. Throughput over Different File Systems in a Data-Center Environment (a) Small Working Sets (b) Large Working Sets

that "hot" files must be given a higher priority in the cache or placed on a faster storage device. Since there is no direct control to increase the priority of caching "hot" files, we try placing the "hot" files on *ramfs* instead.

To show the impact of this hybrid distribution, we designed a test to emulate an ISP provider which services requests for two web sites on the same cluster. For the first web site, we placed all the files in *ext3fs* over an *IDE* disk. For the second web site, we set up two scenarios. In the first scenario, all files are placed in *ext3fs* over an *IDE* disk. In the second scenario, all small files (less than 1MB) are placed in a *ramfs*, and all other files in *ext3fs* over an *IDE* disk. Requests for both web sites are serviced simultaneously. The performance difference of the second website between the two scenarios is our focus.

Figure 7 shows the improvement achieved by the second web site while using *ramfs* to cache the small files. In the figure, we call the workload on the first web site as *background workload* (shown as the x-axis in the figure), and the workload on the second web site as *foreground workload* (shown as the legends in the figure). We can see that the larger the working set of the background workload, the higher the improvement that can be achieved. Compared to placing the small files on *ext3fs*, placing them on *ramfs* achieves a performance improvement of up to 40%. Similarly compared to placing the small files on *PVFS*, placing them on *ramfs* achieves a performance improvement of up to a factor of 4.5. However, the improvement shrinks when the working set size of the foreground workload increases. Further, it is to be noted that the improvement in the *PVFS* based environment is not due to the flushing of the client side cache, but due to the fact that *PVFS* has no client side cache.

5 I/O Requirements

Due to the increasing working set size in current web sites, it might not be possible to fit all the data in the file system cache. For such large workloads, the file system performance is largely dependent on the performance of disk systems. In this section, we study the impact of three different disk systems on the performance of the data-center: *IDE*, *SCSI*, and *a software RAID system*.

For the I/O experiments in the data-center environment, we used the workloads specified in Table 2 (Section 4).

5.1 Basic Performance of Disk Systems

Table 3 shows the basic sequential read and write bandwidth results. We see that *SCSI* provides an improvement of about 50% compared to *IDE*. *RAID* on the other hand allows parallel access to disks providing an improvement of up to a factor of 2 compared to *SCSI*.

Disk systems	Read (MB/s)	Write (MB/s)
IDE	35	39
SCSI	58	55
RAID	120	147

Table 3. Basic Performance of Different Disk Systems

5.2 Impact of Disk Systems in Data-Center

The impact of disk systems in the data-center environment depends on several factors, including the file system cache

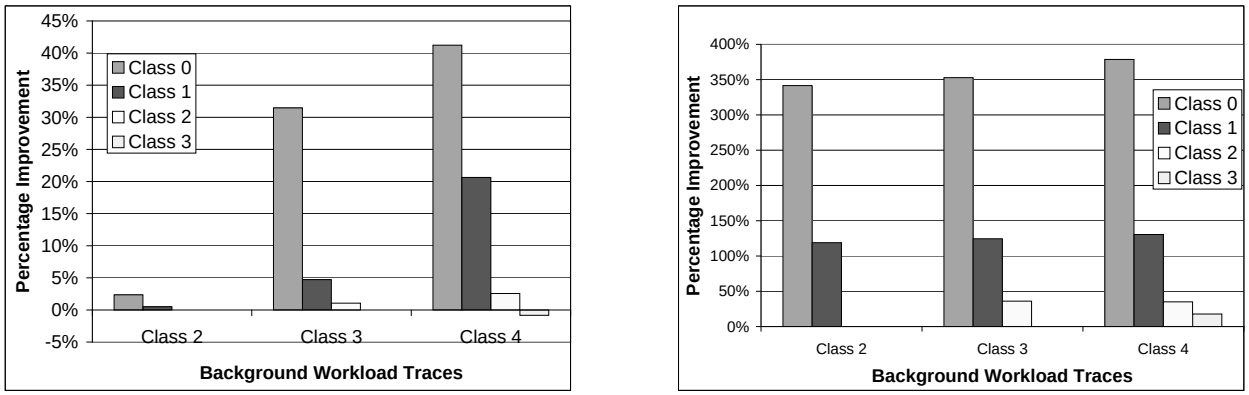


Figure 7. Impact of Hybrid File Distribution in A Multiple Website Environment (a) *ramfs* **with** *ext3fs* **(b)** *ramfs* **with** *pvfs*

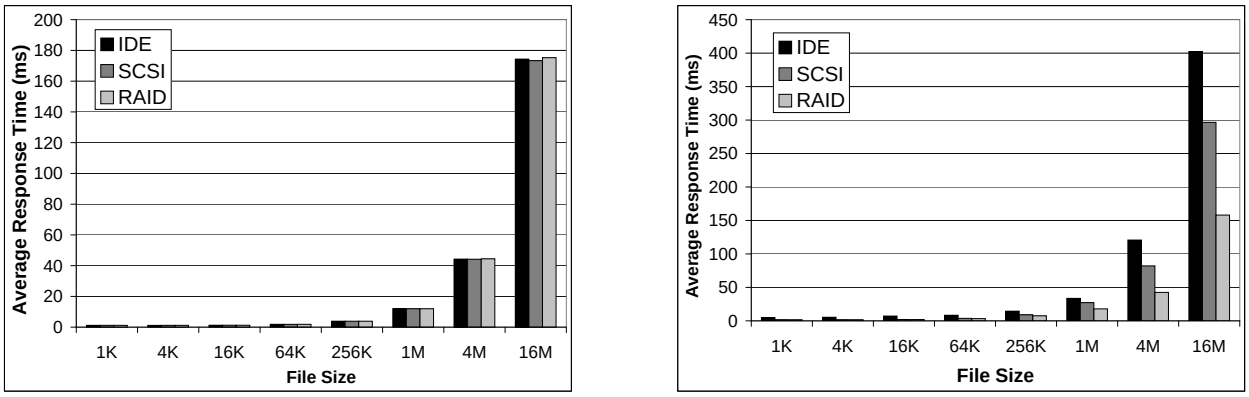


Figure 8. Comparison between IDE, SCSI and RAID systems: (a) File in Cache (b) File not in Cache

hit ratio and the working set size. It also depends on how we utilize the multiple disks present in the system.

5.2.1 Impact of Cache in Data-Center

A miss in the file system cache incurs a disk access. Figure 8a shows the average client response time achieved by web servers with IDE, SCSI and Software RAID disk systems when all files are not in file system cache. The client response time seen is comparable with all three disks for small file sizes. However for large file sizes, Software RAID performs better than SCSI and IDE. Figure 8b shows the average client response time achieved by web servers with IDE, SCSI disks and Software RAID when files are in cache. Since the files are already in cache, all three perform equally for increasing file sizes.

5.2.2 Impact of Cache Hit Ratio

The impact of disk systems varies with different cache hit ratios. We carried out an experiment to characterize this impact. Three types of files with sizes of 4 KB, 64 KB and 1 MB respectively are used. In each case, we control the cache hit ratio manually by accessing different files of similar sizes. The average client response times to access these files are reported in Figure 9.

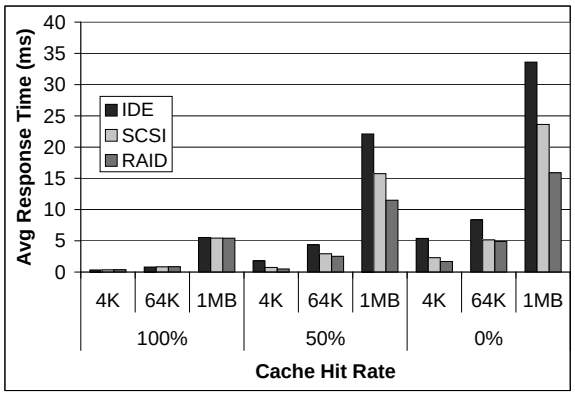


Figure 9. Impact of Cache Hit Ratio

As shown in Figure 9, when the cache hit ratio is 100%, all disk systems show similar performance. When the cache hit ratio is varied from 100% to 50% to 0%, the end response time seen by the user tends to separate out, following the raw performances of the disk systems.

5.2.3 Impact of Working Set Size

Another factor that affects performance is the working set size of files in each workload. As shown in Table 2, the workloads we study have different working set sizes. Figure 10 shows their response times and average throughputs over different disk systems. We do not see an improvement in the response time for workloads with small working set sizes. This is because all the files get cached and there is little or no disk access. However for large working set sizes (such as the class 3 workloads), we see some improvement in the response time due to the fact that the disk access time is increased. In Figure 10b we see that the throughput improvement for *SCSI* and *software RAID* over *IDE* disks increases as the working set size increases. For class 0, the throughput improvements for *SCSI* and *software RAID* disk systems are 6% and 10% respectively. However, when the working set size increases (*Class 3*), we observe that the throughput improvement achieved by *SCSI* and *software RAID* go up to 40% and 100% respectively.

In order to get an in-depth understanding of the disk activity in the above tests, we further looked at the performance of each file. For the class 0 workload, as shown in Figure 11a, since all files can be cached and the number of accesses to each file is relatively large, the average response time of each file is more or less the same. For the class 1 workload (Figure 11b), though all files can be cached, since the number of files increases, the average number of accesses to each file decreases (particularly for large files), making the time for the first time accesses to these files significant (cold start misses). So, the average response time of each file is still sensitive to the disk system. We can a similar trend for the Class 2 workload (Figure 12a). For larger workloads (Figure 12b), the performance improvement is purely due to the disk seek time since such workloads are too large to fit in the file system cache.

5.2.4 Impact of File distribution on disks

As mentioned earlier, RAID over SCSI achieves a higher performance compared to a single SCSI disk. However, it is not very clear whether this overall improvement is due to the parallelization of access for each file (RAID stripes a file over multiple disks for parallel disk access) or because of the fact that the amount of I/O for each disk is lesser (in our testbed, we used three disks per node to form the RAID disk system).

To understand this, in this section, we analyze the performance of the normal SCSI based disk system using the same number of disks as the RAID disk system. The files are distributed over these disks in various fashions as described in Table 4. We also compare the performance of these distributions with the normal SCSI based disk system (with one disk on every node) and that of the RAID disk system.

In the first strategy (*Random*), the files are randomly distributed on the disks. In the second strategy (*Popular*), we sort the files based on popularity in an increasing order and place them on the three disks in a round-robin fashion. In the third strategy (*Size*), we follow a similar approach as the

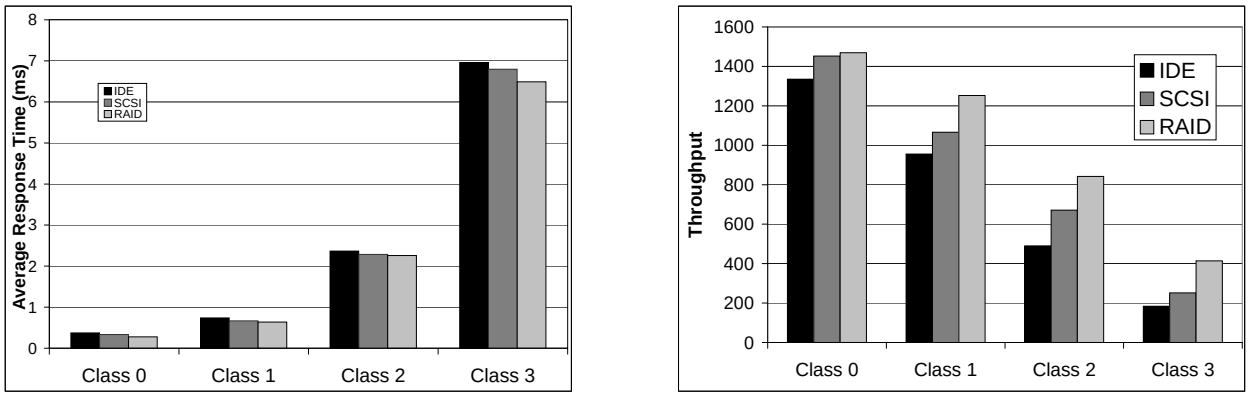


Figure 10. Comparison between IDE, SCSI and RAID systems: (a) Response Time (b) Throughput

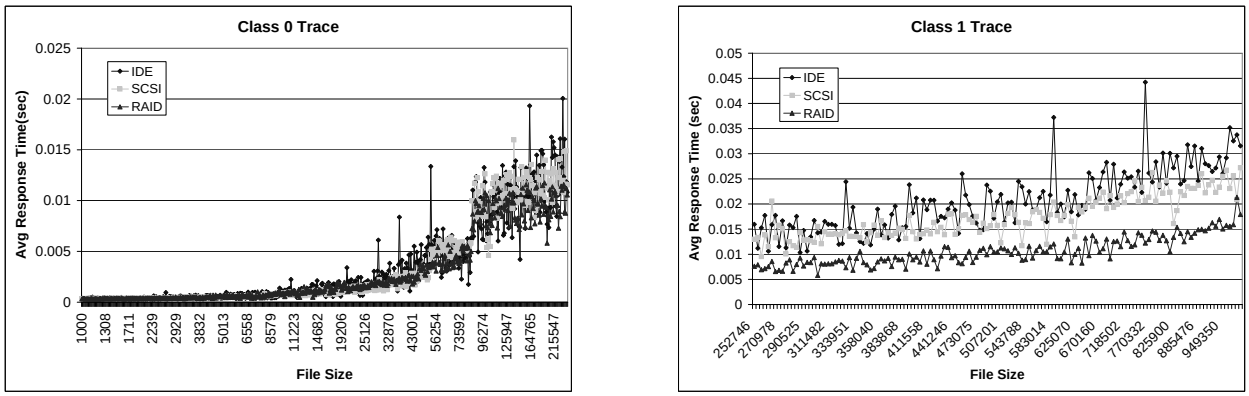


Figure 11. Comparison of IDE vs SCSI vs software RAID: (a) Class 0 Zipf Trace, (b) Class 1 Zipf Trace

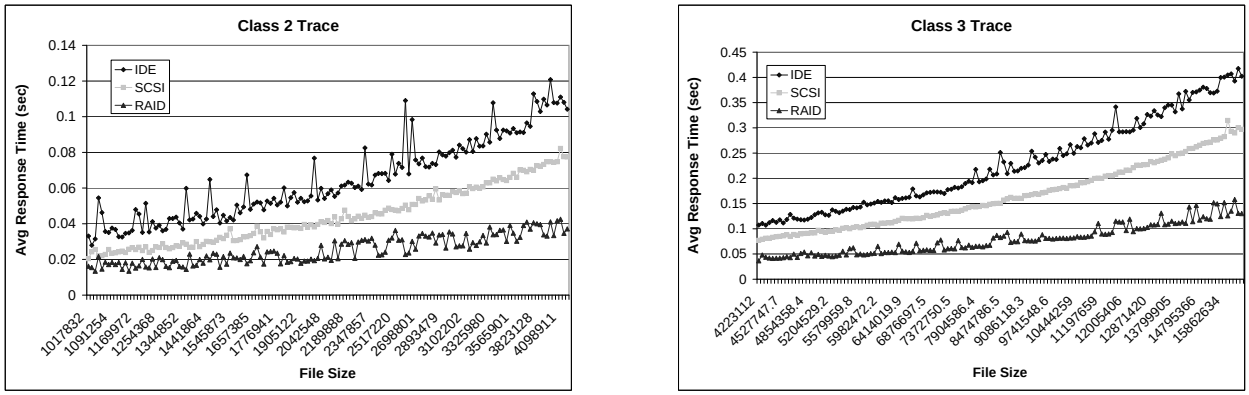


Figure 12. Comparison of *IDE* vs *SCSI* vs *RAID*: (a) Class 2 Zipf Trace, (b) Class 3 Zipf Trace

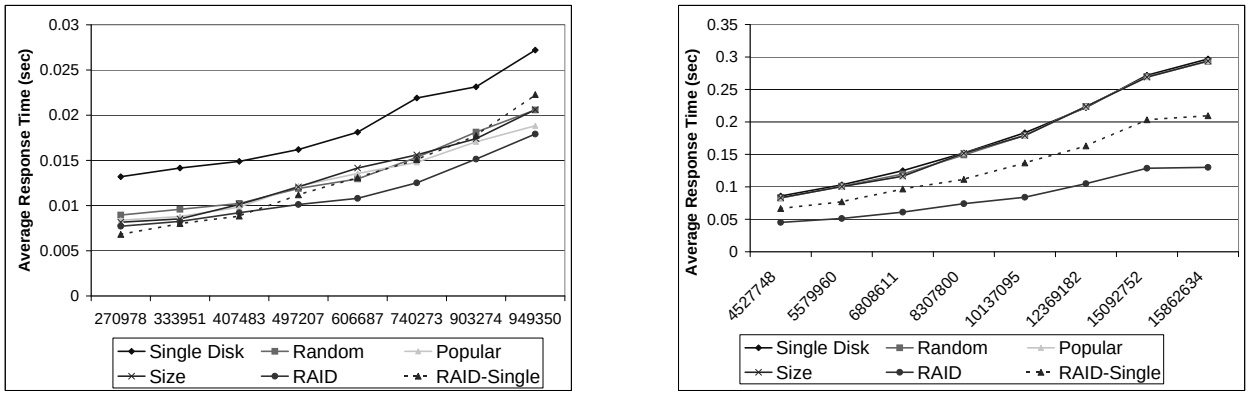


Figure 13. Impact of File Distribution on Multiple Disks: (a) Small Working Set (Class 1), (b) Large Working Set (Class 3)

Distribution	Description
Single Disk	Distribute all the files on single disk on three servers
Random	Randomly distribute the files on multiple disks on three servers
Popularity	Sort the files based on popularity and distribute on multiple disks on three servers
Size-based	Sort the files based on file size and distribute on multiple disks on three servers
RAID-Single	Stripe each file on multiple disks on one server
RAID	Stripe each file on multiple disks on three servers

Table 4. File Distribution on disks

second strategy, except that the files are sorted according to size. We compare these three distribution strategies with the *software RAID* and the single *SCSI* disk cases. The response times for various workloads are reported in Figure 13.

Interestingly, we see that in Figure 13a, for *Class 1* workloads, distributing the files randomly, based on popularity or size gives us a comparable performance to that of the *software RAID*. This points to the fact that most of the performance for these classes of workloads is because of the reduced per disk I/O rather than the parallelization of disk I/O. Internet Service Providers can use this approach to distribute files among various disks and increase their performance.

In Figure 13b, for Class 3 workload, we see that all three distributions perform more or less the same. *Software RAID* does the best since large files get accessed. All three distributions give comparable performance to the single disk case. However for *RAID-Single* case for *Class 3* workloads, using a *Software RAID* with only one server gives significant performance improvement. Increasing the number of the servers in this case also leads to significant improvement. Based on this results, we can say that for these workloads, most of the benefit is achieved from the parallelization of disk access rather than the reduced per disk I/O. This shows that using the same number of disks in a parallel fashion (using a software RAID) can provide a significantly higher performance. ISPs which host such workloads can take advantage of this characteristic by reorganizing their disks to provide parallel access.

6 Network Processing Requirements

In this section, we study the impact of partial offloads of network processing requirements on the network adapter would have in the data-center environment. We will first look at the micro-benchmark results in the form of the base latency and bandwidth achievable by TCP/IP over Myrinet. Next, we will analyze the response time and throughput in the data-center environment.

We also focus on the transmit and receive side processing overheads of the host based TCP/IP stack. We perform detailed measurements to understand the characteristics of TCP/IP processing in the data-center environment and analyze the potential benefits of protocol offloads.

6.1 Performance of TCP/IP

In this section, we compare the basic TCP/IP performance over Myrinet on several platforms that have different CPU speeds and I/O bus speeds.

Figure 14a shows the one-way latency achieved under the different platforms for various message sizes. On Cluster 1, TCP/IP achieves a latency of around 27μs for 4 byte mes-

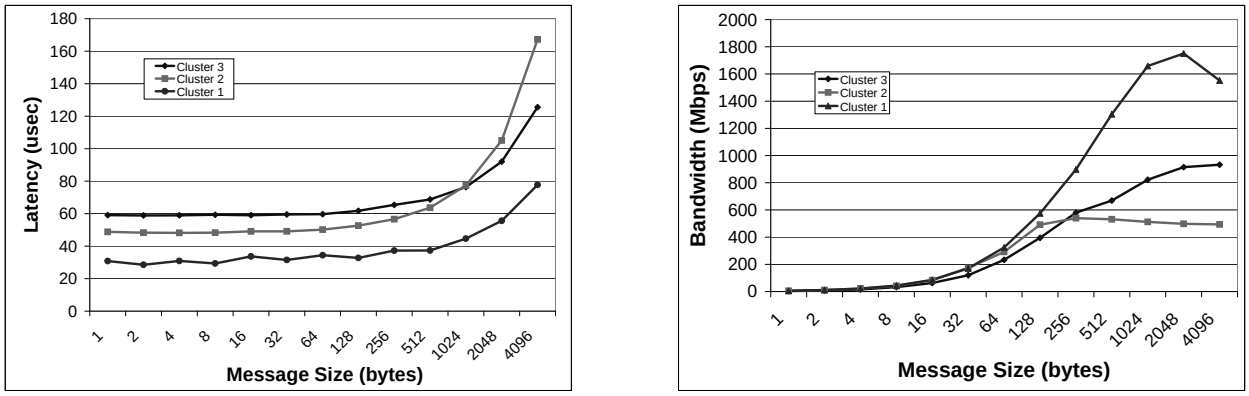


Figure 14. Micro-Benchmarks (Homogeneous Machines): (a) Latency, (b) Bandwidth

sages compared to the 45μs achieved on Cluster 2 and the 59μs achieved on Cluster 3. With increasing message sizes, the difference between the latency achieved by all three clusters tends to increase. For message sizes greater than 1KB, Cluster 3 (700 MHz CPU) performs better than Cluster 2 (1 GHz CPU). This is attributed to the I/O bus speed. Cluster 3 has a 66 MHz PCI compared to the 33MHz PCI on Cluster 2. This shows that for large message sizes, the PCI bus speed is a very critical factor that decides the network performance.

Figure 14b compares the bandwidth achieved by TCP/IP on different platforms. Cluster 1 achieves a peak throughput of 1748 Mbps compared to a 500 Mbps achieved by Cluster 2 and 914 Mbps achieved by Cluster 3. Again, Cluster 2 achieves a lower bandwidth compared to Cluster 3 due to the slow PCI bus.

6.2 Impact of TCP/IP in Data Center

In this section, we measure the response time and throughput achieved by TCP/IP in the data-center environment with Zipf based traces, varying the access pattern. The access pattern is varied with the α coefficient. A high value of α represents a trace with a high temporal locality, i.e., most of the requests are for a small fraction of the documents. Similarly, a low value of α represents a trace with a low temporal locality.

Figure 15 shows the response time and throughput measured on the client side varying the access pattern value of Zipf. As expected, Cluster 1 gives the best performance compared to other two. Cluster 3 achieves better performance than Cluster 2 because of the better PCI bus speed, as we have mentioned earlier.

We can observe that the performance difference is inversely proportional to the access pattern value. The reason is that, in case of a large access pattern value, most data that the clients request is in the proxy tier because requests of clients concentrate on a specific hot set of files. On the other hand, in case of a small access pattern value, we cannot take much advantage of the caching at the proxy tier and most requests have to be passed to the web server tier. Therefore, more communication is realized and the difference of response time and throughput between the three machines becomes larger. In other words, if the cache hit ratio on the proxy tier in the data-center is low, the response time and throughput is largely affected by the intra-cluster network performance of the data-center.

6.3 Checksum Computation Offload

In this section, we study the potential benefits of the checksum processing offload in the data-center environment.

As mentioned earlier, TCP/IP has two kinds of overheads in the message passing path: (i) per-byte overheads and (ii) per-packet overheads. Checksum and Copy are the two main per-byte overheads associated with the TCP/IP stack. In this paper we only consider the checksum offloading because the data copy offloading requires applications to be modified [15]. Since the Myrinet NIC provides a DMA engine that has the ability to perform checksum computation, we try to estimate the benefits attainable by utilizing this to implement the checksum offloading. We analyzed the TCP/IP stack in linux in detail and measured the various components of the per-byte overheads.

6.3.1 Per-byte Operation Overhead

Table 5 shows the per-byte operations overhead for each 1 KB data size on Cluster 3. On the sender side, to maximize the cache effect, Linux performs checksum computation and data copy simultaneously (Jacobson optimization [11]), so we could not separate them. In order to understand the results presented in the table, we describe the path taken by the data in the two cases: (i) with no checksum

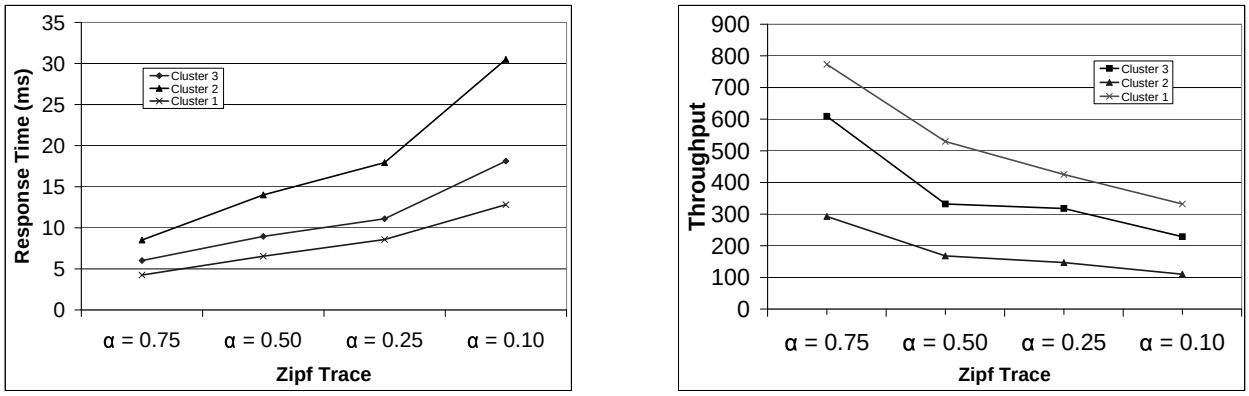


Figure 15. Zipf Trace (a) Response Time (b) Throughput

offload where the host performs the checksum computation and verification and (ii) with the checksum computation and verification offloaded.

In the first case, when the checksum is performed by the host itself, on the sender side both the checksum and copy are integrated. On the receiver side, these are performed separately. However, the interesting thing to note is that when the checksum is performed, the socket buffer is fetched to cache and the copy of the data to the user buffer takes place from cache. However, when the checksum is offloaded, the data is not present in cache (the corresponding cache lines would be invalidated by the memory I/O controller to allow the DMA operation to proceed to memory). This means that the copy of the data to the user buffer takes place from out of cache, increasing its overall cost. This is the difference we observe in the performance of copy with and without checksum offload in Table 5. Also, on the sender side, the checksum is always performed with the buffer in cache, while for the receiver the data is never in cache since it is performed directly after the DMA operation (which requires invalidation of the cache lines by the memory I/O controller). This is also reflected in the values observed in the table.

Environment	Perbyte Operations	Sender	Receiver
Microbenchmark	Checksum	3.69	8.42
Microbenchmark	Copy w/o Cks Off		1.95
Microbenchmark	Copy w/ Cks Off	1.95	8.40
Data-center	Checksum	3.69	8.42
Data-center	Copy w/o Cks Off		5.17
Data-center	Copy w/ Cks Off	3.65	8.40

Table 5. Per-byte Operation Overhead for 1KB data Size on Sender and Receiver in Micro-benchmark and Data-Center Environments

In Table 5, we have distinguished between the micro-benchmark tests and the data-center based tests. Micro-benchmark tests utilize the same buffer for every measurement iteration, and there is no other process running on the system. Therefore, we can maximize the processor cache effect during the test. However, offloading the checksum to the network adapter would maximize these cache misses and would lead to minimal performance benefits in this case. Further, it is to be noted that this is the pure communication micro-benchmark test. In an environment such as the data-center, where this communication is only a part of the overall performance, the benefit can be expected to be even lesser. Figure 16 shows the estimated performance of the pure micro-benchmark test with checksum offloading. This result is in accordance with the previous studies on checksum offload engines [14].

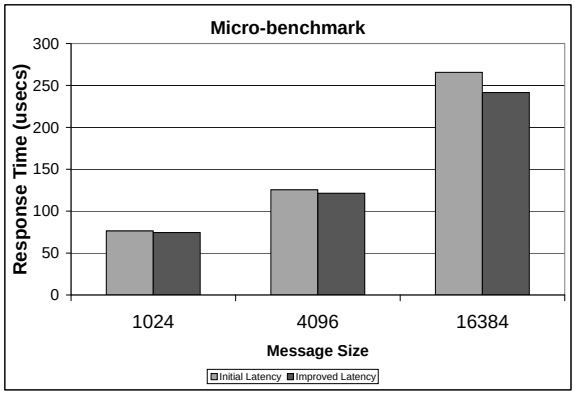


Figure 16. Checksum Offload benefits in Micro-Benchmark Tests

However, when we have a look at the difference between the copy with checksum offload and without checksum offload costs in Table 5, we see that the difference between

these two values is significantly lesser for the data-center environment as compared to the micro-benchmark tests. This suggests that in the data-center environment, the number of cache misses might be higher than that in the micro-benchmark test.

To verify this, we measured the number of L2 cache misses[2] occurring in the micro-benchmark test as well as the data-center test (Figure 17a) using the Pentium Performance Measurement Counters (PMCs). We can see that the number of cache misses is significantly higher for the data-center test compared to the micro-benchmark test. This shows that by offloading the checksum computation to the network adapter, we will not be introducing many additional cache misses; and hence can expect some benefit for checksum offload engines in the data-center environment.

Figure 17b shows the estimated benefits for checksum offload in the data-center environment. We can see that checksum offload engines provide a benefit of up to 15% for a 16KB message. It is to be noted that this is the improvement in the end-user performance. The actual improvement in the communication part of the test can be expected to be much higher than this.

6.3.2 Per-packet Operation Overhead

As discussed in the previous section, per-packet overhead includes the overhead associated with the TCP/IP protocol, allocation and release of memory buffers and interrupts from the network device for packet arrival and transmission. Since TCP/IP protocol is complex, it is very difficult to implement all per-packet processing onto the hardware. On the other hand, if we offload them to firmware, we cannot expect a benefit. Therefore, the policy that offloads a part of per-packet processing into NIC and implements a hardware for a large portion of it would be beneficial. Figure 18 shows the estimated benefits of offloading the per-packet overhead in the form of partial or complete TCP offload engines (partly on the firmware and partly on the hardware (H/w), shown as the legend in the figure). We see that the TCP offload engines provide benefits only when there is significant offloading to the hardware (ASIC based). This is because protocol offload onto the firmware increases the per-packet overhead mainly due low CPU speed at the firmware (the host CPU speed is 700 MHz and firmware speed is 132 MHz in this experiment).

7 Compute Requirements

In this section, we study the implications of varying the compute resources on the overall performance of the data-center. In particular, we study two scenarios: (i) varying the host CPU speed and (ii) varying the number of host CPUs. We first present basic data-center level response time and throughput measurements and then follow it up with more detailed benchmarks to understand the implications of varying the compute resources.

[2]The values presented are actually the number of cache lines being fetched to L2 and not exactly the L2 cache misses.

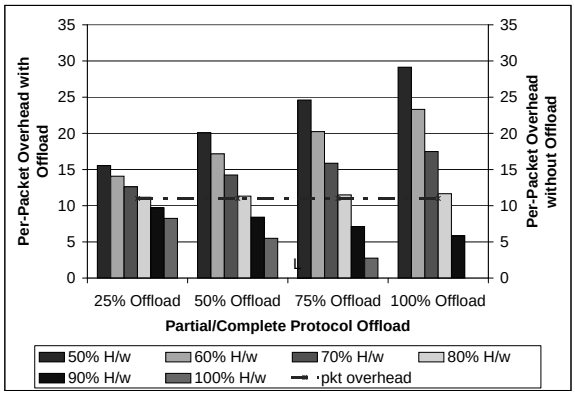


Figure 18. Estimated benefits of Partial/Complete Offload on per-packet Overhead

7.1 Data-Center Response Time and Throughput Measurements

In this section, we discuss the base performance of the data-center with static and dynamic content.

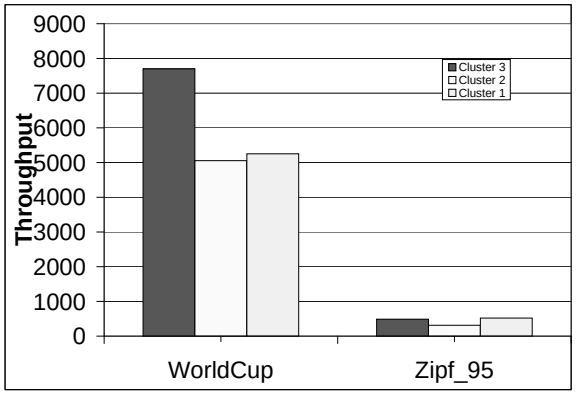
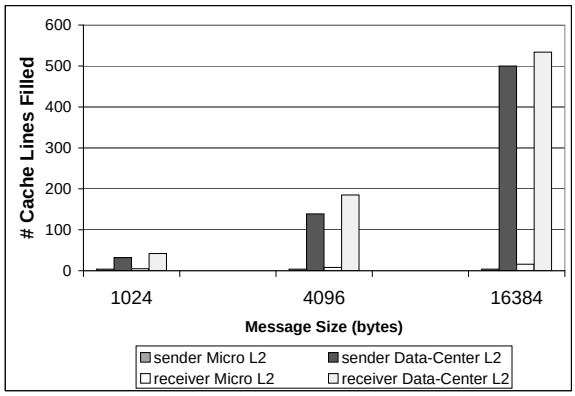


Figure 22. Throughput in a Worldcup trace

Figure 19a shows the response times seen by the client for various average file sizes. As expected, Cluster 1 gives a faster response time compared to the other two due to a better CPU speed (2.4 GHz). Further, for small file sizes, we see that all clusters give equal performance. Figure 19b shows the client response time for dynamic content. As seen in Figure 19a, Cluster 1 does better than Clusters 2 and 3.

Figure 20a shows the throughput seen at the client side for



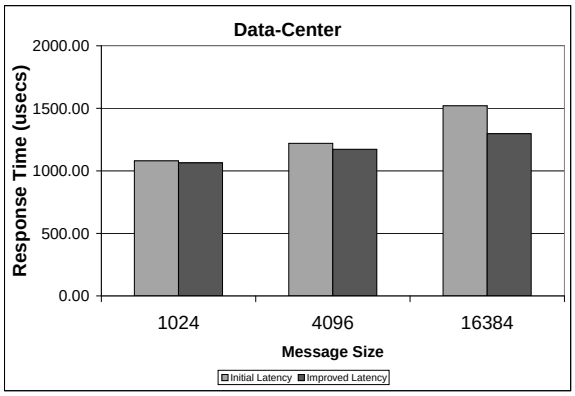
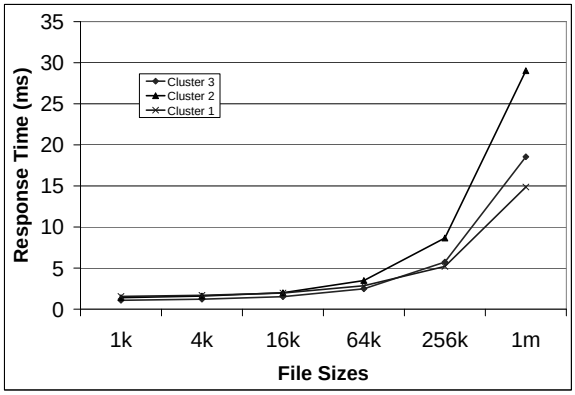


Figure 17. (a) Cache Lines Filled during copy operation (b) Improved Data-Center Response Time with Checksum Offload



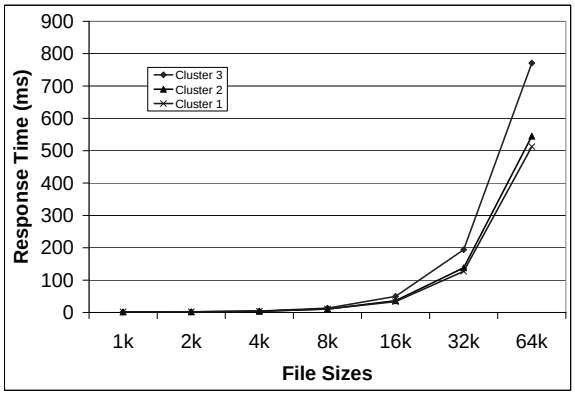
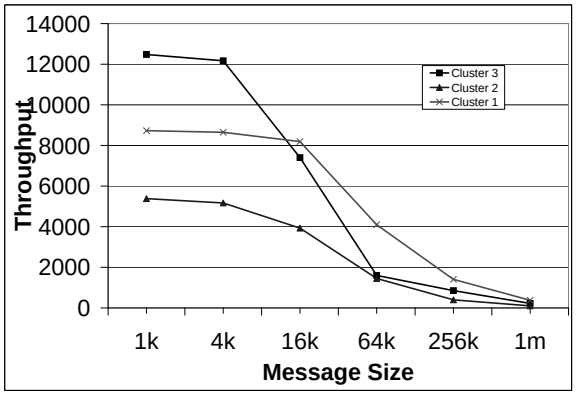


Figure 19. Data-Center Response Time (a) Static Content (b) Dynamic Content



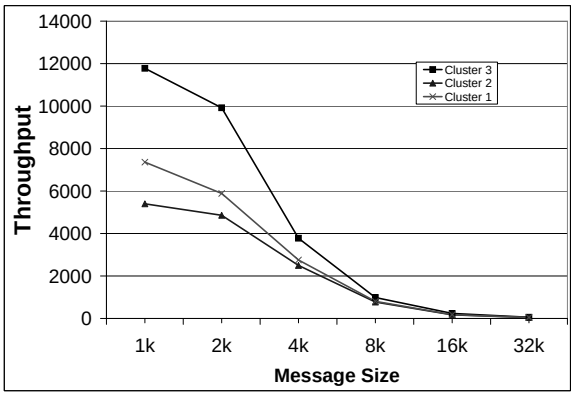


Figure 20. Data-Center Throughput (a) Static Content (b) Dynamic Content

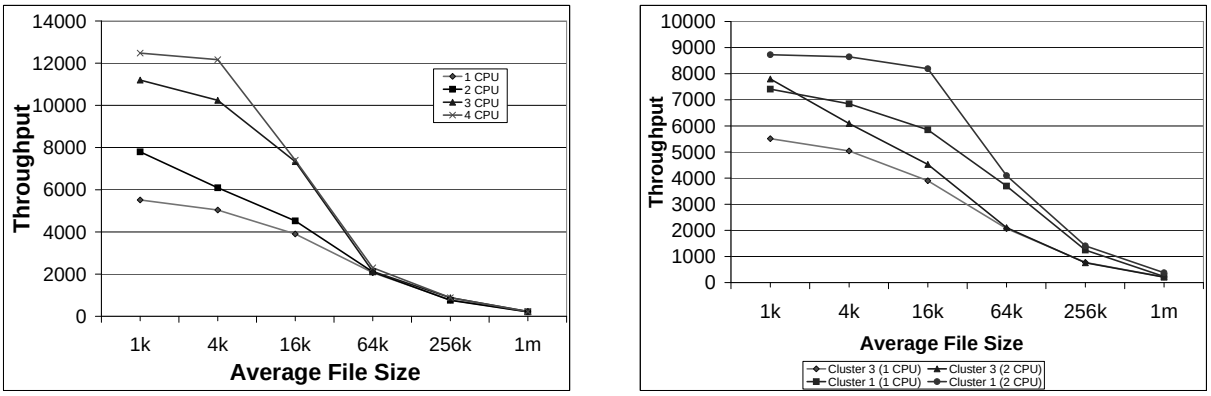


Figure 21. Throughput variation (a) Number of CPUs (b) CPU Speed

static content. Surprisingly, for small file sizes slower machines with more CPUs perform better compared to faster machines with fewer CPUs! This result was initially puzzling, but the reason soon became apparent. Web workloads typically tend to have portions of compute and I/O interleaved. This characteristic of web requests makes the CPU speed a less critical aspect for the overall performance of the data-center. On the other hand, having more CPUs allows more threads to run in a parallel fashion resulting in a higher throughput. For the response time test, since there is only one request in the data-center at any point of time, the node with more CPUs is not able to allow a higher concurrency in the access. On the other hand, a faster CPU can perform better for the intermittent compute parts in the overall request to show a better response time in Figure 19a.

For large file size requests, however, faster machines tend to be better than the slower machines. This is because the I/O part for large messages becomes significantly larger than the compute part resulting in an imbalance in the interleaving of the compute and I/O parts. We see a similar trend for real traces in Figure 22.

Figure 20 shows the throughput achieved by the three clusters for dynamic content. The compute intensive nature of typical dynamic content ensures a high compute to I/O interleaving even for large messages. It is to be noted that in our experiments, the amount of computation was set to be proportional to the file size in order to have a more deterministic value for the amount of computation involved with each request.

7.2 Impact of CPU in a multi-tier data-center

In order to verify the impact of multiple CPUs on the overall performance of the data-center, we performed two experiments. In the first experiment, we varied the number of CPUs on the same machine to see the actual improvement in the performance per CPU. The second experiment shows the performance of two machines with the same number of CPUs but with different processing speeds.

In order to reduce the number of CPUs present in a system, a kernel module was written so that it can reschedule a job on a particular processor every time the process gets swapped out. We spawned all apache threads on different CPUs depending on the experiments. Figure 21a shows the throughput seen in the multi-tier data-center with varying number of CPUs. This figure shows that as the number of CPUs increase, the throughput obtained increases linearly for small message sizes. However for large file sizes, since I/O becomes the bottleneck, increasing the number of CPUs does not improve performance. In Figure 21b we compare the throughput for different clusters with different CPU speeds and varying number of CPUs. One interesting observation is that for workloads with small file sizes we see a significant performance improvement with a slower machine with more number of CPUs than a faster machine with fewer CPUs. This result again re-verifies the observation we made in Section 7.1. This shows that ISPs which host such workloads can take advantage of this characteristic by adding more processors in their environment and thereby increasing their availability rather than trying to upgrade the processing speed as such.

8 Conclusions and Future Work

The phenomenal growth and popularity of cluster-based multi-tier data-centers has not been accompanied by a system-wide understanding of the various resources and their deployment strategies. Due to the large variation in the kind of workloads present in today's data-center environments, there's no study which characterizes these workloads with respect to the architectural requirements they have. In other words, the implications of the various system resources such as CPU, file system, disk, network, I/O on these different kinds of workloads have not been previously

studied.

This paper presents a detailed analysis of the various architectural components in a cluster-based multi-tier data-center environment and their influence on each tier in general and the overall performance of the data-center in particular. Specifically, we focused on resources associated with the file system (*ext3fs*, *ramfs*, and *PVFS*), I/O (*IDE*, *SCSI*, and *software RAID*), the TCP/IP protocol stack and the compute resources (number of CPUs and CPU speed) in the data-center. We evaluated these resources with micro-benchmarks and studied their impact in a multi-tier data-center environment. Together with studying the implications of each of these resources on the performance of the data-center, we have also suggested various approaches in which data-center administrators can maximize their performance based on the resources they have.

As a future work, we propose to develop an analytical model for multi-tier data-centers and evaluate it using the above mentioned workloads.

9 Related Work

Recently Boston university [5] developed a tool which gives complete insight of Web Server activity and helps in identifying the performance bottlenecks. Several others have studied the performance of a busy WWW server [1, 21, 23].

Spasojevic [30] suggest using a wide-area file system within the World-Wide Web. Martin F. Arlitt et all [3] have studied workload characteristics that are common to different workloads and emphasized the importance of caching and performance issues in web servers. Also, Jaidev et all [26] have looked at network processing overhead in web servers. They claim that protocol offload would give significant benefits for static workloads (compute-intensive) and not for I/O intensive workloads. However, to the best of our knowledge, our study is unique since we analyze various components of a cluster-based multi-tier data-center in an integrated manner which tightly couples with the expected workload characteristics.

10 Acknowledgments

We would like to thank Sundeep Narravula, Savitha Krishnamoorthy and Sayantan Sur for all the technical support they provided in this paper. We would also like to thank the PVFS team at the Argonne National Laboratory and Clemson University for giving us access to the latest version of the PVFS implementation and for providing us with crucial insights into the implementation details.

References

[1] Jussara Almeida, Virgilio Almeida, and David Yates. Measuring the behavior of a world-wide web server. Technical Report 1996-025, 29, 1996.

[2] The Internet Traffic Archive. http://ita.ee.lbl.gov/html/traces.html.

[3] Martin F. Arlitt and Carey L. Williamson. Web server workload characterization: The search for invariants. In *Measurement and Modeling of Computer Systems*, pages 126–137, 1996.

[4] P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Sockets Direct Protocol over InfiniBand in Clusters: Is it Beneficial? In *the Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, Texas, March 10-12 2004.

[5] Azer Bestavros, Robert L. Carter, Mark E. Crovella, Carlos R. Cunha, Abdelsalam Heddaya, and Sulaiman A. Mirdad. Application-level document caching in the Internet. In *Proceedings of the 2nd International Workshop in Distributed and Networked Environments (IEEE SDNE '95)*, Whistler, British Columbia, 1995.

[6] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM (1)*, pages 126–134, 1999.

[7] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System for Linux Clusters. In *4th Annual Linux Showcase and Conference*. USENIX Association, 2000.

[8] J. Chase, A. Gallatin, and K. Yocum. End system optimizations for high-speed TCP. *IEEE Communications Magazine*, 39(4):68–74, 2001.

[9] A. Ching, A. Choudhary, W. Liao, R. Ross, and W. Gropp. Noncontiguous I/O through PVFS. In *Cluster Computing*, 02.

[10] Guo Chuanxiong and Zheng Shaoren. Analysis and Evaluation of the TCP/IP Protocol Stack of Linux. http://www.ifip.or.at/con2000/icct2000/icct452.pdf.

[11] D. Clark, V. Jacobson, J. Romkey, and H.Salwen. An Analysis of TCP processing overhead. *IEEE Communications*, June 1989.

[12] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM symposium on Communications architectures & protocols*, pages 200–208. ACM Press, 1990.

[13] Aaron Falk and Mark Allman. On the effective evaluation of TCP. *ACM Computer Communication Review*, 5(29), 1999.

[14] Annie Foong, Herbet Hum, Tom Huff, Jaidev Patwardhan, and Greg Regnier. TCP Performance Revisited. In *the Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, 2003.

[15] Andrew Gallatin, Jeff Chase, and Ken Yocum. Trapeze/IP: TCP/IP at near-gigabit speeds. pages 109–120.

[16] Robert Horst. IP Storage and the CPU Consumption Myth.

[17] Jau-Hsiung Huang and Chi-Wen Chen. On Performance Measurements of TCP/IP and its Device Driver. Proc. 17th Annual Local Computer Network Conference, Minneapolis.

[18] Jonathan Kay and Joseph Pasquale. The Importance of Non-Data Touching Processing Overheads in TCP/IP. In Proceedings of the 1993 SIGCOMM, pp. 259–268, San Francisco, CA, September 1993.

[19] Jonathan Kay and Joseph Pasquale. Measurement, analysis, and improvement of UDP/IP throughput for the DECstation 5000. In *USENIX Winter*, pages 249–258, 1993.

[20] Hyok Kim, Hongki Sung, and Hoonbock Lee. Performance Analysis of the TCP/IP Protocol under UNIX Operating Systems for High Performance Computing and Communications. In *the Proceedings of International Conference on High Performance Computing (HPC)*, 1997.

[21] A. Mahanti. Web proxy workload characterisation and modelling, 1999.

[22] Evangelos P. Markatos. Speeding up TCP / IP : Faster processors are not enough. Technical Report 297, 2001.

[23] J. C. Mogul. Network behavior of a busy web server and its clients. Technical Report Technical Report WRL 95/5, DEC Western Research Laboratory, Palo Alto, CA, 1995.

[24] David Mosberger, Larry L. Peterson, Patrick G. Bridges, and Sean O'Malley. Analysis of Techniques to Improve Protocol Processing Latency. Technical report, University of Arizona, 1996.

[25] S. Narravula, P. Balaji, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Supporting Strong Coherency for Active Caches in Multi-Tier Data-Centers over InfiniBand. In *SAN*, 2004.

[26] Jaidev P. Patwardhan, Alvin R. Lebeck, and Daniel J. Sorin. Communication breakdown: Analyzing cpu usage in commericial web workloads. In *IEEE Internation Symposium on Performance Analysis of Systems and Software*, 2004.

[27] Vern Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, pages 277–292, 1997.

[28] Hemal V. Shah, Dave B. Minturn, Annie Foong, Gary L. McAlpine, Rajesh S. Madukkarumukumana, and Greg J. Regnier. CSP: A Novel System Architecture for Scalable Internet and Communication Services. In *the Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, pages pages 61–72, San Francisco, CA, March 2001.

[29] Huseyin Simitci, Chris Malakapalli, and Vamsi Gunturu. Evaluation of SCSI over TCP/IP and SCSI over Fibre Channel Connections.

[30] Mirjana Spasojevic, C. Mic Bowman, and Alfred Spector. Using a wide-area file system within the World Wide Web. 1994.

[31] Evan Speight, Hazim Shafi, and John K. Bennett. WSDLite: A lightweight alternative to windows sockets direct path. pages 113–124.

[32] W. Richard Stevens. *TCP/IP Illustrated, Volume I: The Protocols*. Addison Wesley, 2nd edition, 2000.

[33] R. Thakur, W. Gropp, and E. Lusk. On Implementing MPI-IO Portably and with High Performance. In *the 6th Workshop on I/O in Parallel and Distributed Systems*, 1999.

[34] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, Volume II: The Implementation*. Addison Wesley, 2nd edition, 2000.

[35] Eric Yeh, Herman Chao, Venu Mannem, Joe Gervais, and Bradley Booth. Introduction to TCP/IP Offload Engine (TOE). http://www.10gea.org, May 2002.