Workload-driven Analysis of File Systems in Shared Multi-tier Data-Centers over InfiniBand

K. Vaidyanathan P. Balaji H. -W. Jin D. K. Panda
Computer Science and Engineering
The Ohio State University
{vaidyana, balaji, jinhy, panda}@cse.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. Each tier in a multi-tier data-center has different requirements and behavior. Accordingly, it is a non-trivial problem to analyze the impact of various system resources and their influence on each tier. In addition, typical data-center workloads have a wide range of characteristics. They vary from high to low temporal locality, large documents to small documents, the number of documents and several others. The different characteristics of each kind of workload makes this problem quite challenging. Further, in the past few years several researchers have proposed and configured data-centers providing multiple independent services, known as shared data-centers. The requests for these different services compete with each other while sharing the resources available in data-center, thus further complicating this problem. In this paper, we focus on analyzing the impact of the file system in a shared data-center environment. We study the impact of both local file systems (*ext3fs* and *ramfs*) and network-based file systems (*PVFS* and *Lustre*) in three broad aspects namely: (i) Network Traffic Requirements, (ii) Aggregate cache size and (iii) Cache pollution effects. Based on the insights gained from these broad issues we propose a multi file system data-center environment to utilize each file system only for environments where it is most suited for, thus taking the best capabilities of all the file systems. Our experimental results show that this approach can improve the performance by up to 48% in a shared data-center environment for static (time invariant) workloads showing high temporal locality, up to 15% for static workloads with low temporal locality and up to 40% for dynamic (time variant) workloads.**

I. Introduction

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. On the other hand, 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. Based on these two trends, several researchers have proposed the feasibility and potential of cluster-based multi-tier data-center environments [21], [14], [18], [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. Each of these duplicated programs receive a request from the end user using a higher level protocol such as *HTTP*. Depending on the request, each program can either directly fetch a document from the file system and return it to the user or fetch some raw data from the file system, process it and send the processed output to the user. In either case, the interaction of the duplicated programs with the file system plays an important role in the end performance perceived by the users.

Multi-tier data-centers generally consist of three tiers; the proxy tier, the application/web tier and the database tier. Since each tier has different requirements and behavior, it is a non-trivial problem to analyze the impact of various system resources (such as the file system) and their influence on each tier. In addition, typical data-center workloads have a wide range of characteristics. They vary from high to low temporal locality, large documents to small documents (download sites vs book stores), the number of documents and several others. The different characteristics of each kind of workload makes this problem quite challenging. Further, in the past few years several researchers have proposed and configured data-centers providing multiple independent services, known as shared data-centers [9], [10]. For example, several ISPs and other web service providers host multiple unrelated web-sites on their data-centers. The requests for these different web-sites compete with each other while sharing the resources available in data-center, thus further complicating this problem.

In this paper, we focus on analyzing the impact of the file system in a shared data-center environment. We study the impact of both local file systems such as *ext3fs* and *ramfs* and network-based file systems such as *PVFS* and *Lustre* in three broad aspects:

1) Network Traffic Requirements: Network file systems require data to be fetched over the network on every request to the file system. This has several implications. First, if the amount of data fetched over the network is very high, this might cause a network traffic bottleneck and might hinder with other network operations performed in the data-center environment. Second, obtaining the handle to a file is no longer a local operation; this might cause file opening and closing to be a significantly expensive operation as compared to local file systems. Third, fetching data over the network is beneficial when the data is fetched in large bursts, thus utilizing the bandwidth provided by the network. Fetching small bursts of data might under-utilize the network and might lead to sub-optimal performance.
2) Aggregate cache size: While local file systems have a low cache hit time, they do not have any interaction with the other nodes in the system as far as the file management is concerned. Thus, the documents that need to be served for a web-site (or multiple web-sites) need to

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

be replicated on each server node. While this might not be a concern with respect to disk space for most websites, it might limit the aggregate amount of cached content due to replication of the content on the various nodes. On the other hand, network-based file systems allow the cache to be distributed (or striped) across various nodes, thus getting rid of the replication requirement.

3) Cache pollution effects: Caching has a significant impact on the performance of the web/proxy server. Due to the high frequency of accesses, popular files tend to be highly sensitive to the caching capability of the file system. Ideally, we would like these popular documents to always be cached. However, in a shared data-center environment hosting 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 resulting in several cache misses and a significant drop in performance. As we will see in the later sections, for shared data-center environments, this degradation can be up to *ten* times in some cases.

Based on the insights gained from these broad issues associated with shared data-centers, in this paper we propose a multi file system data-center environment. This approach attempts to handle the above mentioned issues by utilizing each file system only for environments where it is most suited for, thus taking the best capabilities of all file systems. Our experimental results show that this approach can improve the performance by up to 48% in a shared data-center environment for static (time invariant) workloads showing high temporal locality, up to 15% for static workloads with low temporal locality and up to 40% for dynamic (time variant) workloads.

The remaining part of the paper is organized as follows: Section II provides a brief background about multi-tier data-centers, the Parallel Virtual File System (PVFS) and the Lustre File System. In Section III we mention the workload and testbed that we used. In Section IV, we evaluate the different file system in the data-center environment using different workloads and provide several solutions that allow the design of next generation data-centers to be tightly coupled with the expected workload characteristics. We conclude the paper in Section VI.

II. Background

In this section, we give a brief overview of the architecture of multi-tier data-centers, the Parallel Virtual File System (*PVFS*) and the Lustre File System.

A. Data-Center Tiered Architecture

A typical data-center architecture consists of multiple tightly interacting layers known as tiers. Each tier can contain multiple physical nodes. Requests from clients are load-balanced on to the nodes in the proxy tier. This tier mainly does caching of content generated by the other back-end tiers, balances the requests sent to back-end servers and other such services.

The second tier consists of two kinds of servers. First, those which host static content such as documents, images and others which do not change with time are referred to as web-servers. Second, those which compute results based on the query itself and return the computed data in the form of a static document to the users. These servers, referred to as application servers, usually handle compute intensive queries which involve transaction processing and implement the data-center business logic.

The last tier consists of database servers. These servers hold a persistent state of the databases and other data repositories. For simple queries, such as search queries, etc., these servers tend to be more I/O intensive requiring a number of fields in the database to be fetched into memory for the search to be performed. For more complex queries, such as those which involve joins or sorting of tables, these servers tend to be more compute intensive.

Other than these three tiers, various data-center models specify multiple other tiers which either play a supporting role to these tiers or provide new functionalities to the data-center. For example, the CSP architecture [21] specifies an additional edge service tier which handles security, caching, SAN enclosure of packets for TCP termination and several others.

Shared Data-Centers: A clustered data-center environment essentially tries to utilize the benefits of a cluster environment (e.g., high performance-to-cost ratio) to provide the services requested in a data-center environment. As mentioned earlier, researchers have proposed and configured data-centers to provide multiple independent services, such as hosting multiple web-sites, forming what is known as shared data-centers. For example, several service providers host multiple websites in their data-centers. Hosting multiple services or websites in a single data-center environment has interesting implications on the caching capabilities of the data-center.

B. Parallel Virtual File System (PVFS)

PVFS [8] is a parallel cluster-based file system. It was designed to meet the increasing I/O demands of parallel applications. A number of nodes in the cluster system can be configured as I/O servers and one of them (either on I/O server node or on a different node) as a metadata manager. Figure 1 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.

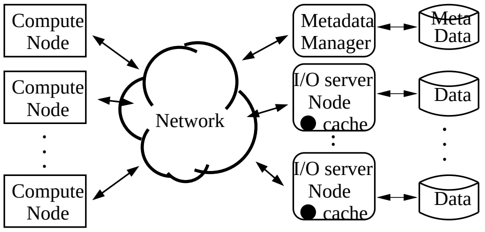

Fig. 1. A Typical PVFS Setup

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. A manager daemon running on a metadata manager node handles metadata operations like file permissions, 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 [11]. PVFS can be used with multiple APIs: a native PVFS API, the UNIX/POSIX API, MPI-IO [23], 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.

C. Lustre File System

Lustre [12] is an open source, high-performance cluster file system designed to eliminate the problems of performance, availability, scalability in distributed systems. Lustre uses object based disks for storage and metadata servers (MDS) for storing the metadata. Distributed Object Storage Targets (OSTs) are responsible for the actual file I/O. Figure 2 demonstrates a typical *Lustre* environment. As shown in Figure, apart from increased network bandwidth, *Lustre* also supports client-side caching. On the other hand, *PVFS* does not support this feature and we will see the impact of client-side caching especially for database workloads and its benefits compared the *PVFS* in later sections.

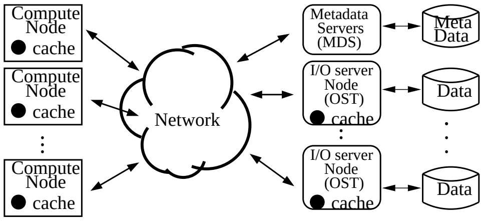

Fig. 2. A Typical Lustre Setup

Lustre leverages open standards such as Linux, XML, SNMP, readily available source libraries and existing file systems to provide scalable, reliable and powerful distributed file system. Lustre maximizes the performance and productivity by using sophisticated fail-over, recovery and replication techniques to eliminate downtime and maximize file system availability. In addition, Lustre also supports strong file and metadata locking semantics to maintain coherency of the file system.

III. Data-center Requirements

Typical data-center workloads have a wide range of characteristics. These range of characteristics, coupled with the requirements and behavior of exclusive as well as shared data-centers, makes analyzing the impact of the various components in the data-center such as the file system, a non-trivial problem. In Section III-A, we discuss the broad characteristics of different kinds of workloads. In Section III-B, we discuss the implications of the different kinds of file systems on the exclusive as well as shared data-centers. Experimental analysis of these implications is presented in Section IV.

A. Workload

Due to the varying characteristics of workloads, in this paper, we classify the workloads in four broad categories: (i) Single-file Micro workloads, (ii) Zipf-like workloads and (iii) Dynamic content workloads.

Single-File Micro workloads: This workload contains only a single file. This workload is used to study the basic performance achieved by the data-center environment for different file systems without being diluted by other interactions in more complex workloads. 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.

Zipf-like Workloads: It has been well acknowledged in the community that most workloads for data-centers hosting static content, follow a Zipf-like distribution [7]. According to Zipf law, the relative probability of a request for the i'th most popular document is proportional to $1/i^{\alpha}$, where α determines the randomness of file accesses. In our experiments, we use workloads which have constant α value but vary the working set size and also a constant working set size varying the α value.

Dynamic Content Workloads: We used three kinds of dynamic content workloads: (i) A Transactional web benchmark (TPC-W) to emulate the operations of an e-commerce website, (ii) RUBiS benchmark to represent auction sites [22] modeled after eBay.com and (iii) RUBBoS benchmark to represent bulletin board systems [6] modeled after slashdot.org.

B. File System Implications on Data-Centers

Figure 3 shows the interaction of the data-center with the file system. For local file systems, the servers can serve the content locally without requiring any interaction with other nodes as far as the file management is concerned. On the other hand, in a network-based file system, the servers have to contact one or more of the file system servers over the network to retrieve the file.

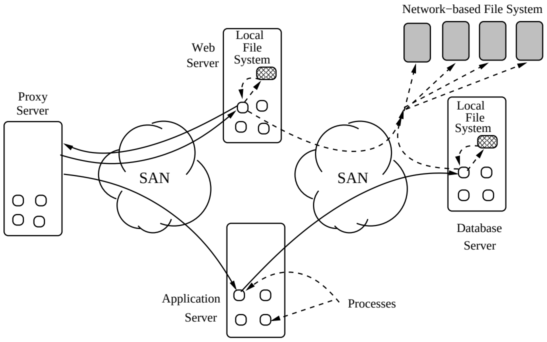

Fig. 3. Data-Center and File System Interaction

In this section, we discuss three broad aspects associated with the interaction of the various workloads with the file system component in the data-center environment, namely: (i) Network Traffic Requirements, (ii) Aggregate Cache Size and (iii) Cache pollution effects.

1) Network Traffic Requirements: Network file systems require data to be fetched over the network on every request to the file system. This has several implications.

Amount of data fetched: If the amount of data fetched over the network is very high, this might cause a network traffic bottleneck and might hinder with other network operations performed in the data-center environment. For static content (e.g., html pages, etc), the data transfer path is quite straight forward. The file requested by the clients is directly present on the file system; it needs to be retrieved from the file system and sent to the end client. Thus, we can derive a straight forward correlation between the amount of data requested by the client and the total amount of network traffic this request would generate (including the transfer of the file to the client and the retrieval of the file from the file system for network-based file systems).

For dynamic content (e.g., output generated by CGI scripts, Java servlets, etc), the data transfer path is more complicated. For example, in transactional workloads (such as TPC-W), the database needs to retrieve and access several objects in the database from the file system. Once this raw data is retrieved, it needs to be processed and the processed output is sent to the client. Thus, there is no straight forward correlation between the amount of data retrieved from the file system and the amount of data sent to the client.

Metadata operations: For network-based file systems, metadata operations on files such as obtaining the handle to read a file are no longer a local operation. For example, *PVFS* has a manager daemon that handles metadata operations like file create, open, close, and remove operations. This might cause file opening and closing to be a significantly expensive operation as compared to local file systems. The manager, however, does not participate in read/write operations; the client library and the I/O daemons handle all file I/O without the intervention of the manager.

Network Utilization: Fetching data over the network is beneficial when the data is fetched in large bursts, thus utilizing the bandwidth provided by the network. Fetching small bursts of data might under-utilize the network and might lead to sub-optimal performance. For dynamic content, since the retrieval of raw data from the file system is accompanied by processing of this data, it might not be done in a single burst of data read, but rather as multiple reads of small data bursts.

2) Aggregate Cache Size: As mentioned earlier, while local file systems have a low cache hit time, they do not have any interaction with the other nodes in the system as far as the file management is concerned. Thus, if three servers are assigned to service requests coming to a certain website, each of these servers is completely unaware of the existence of the other servers. Accordingly, all required documents are made available locally to the server. Further, since the servers are unaware of each other, the documents that need to be served for the web-site need to be replicated on each server node. While this might not be a concern with respect to disk space used for most websites, it might limit the aggregate amount of cached content due to replication of the content on the various nodes.

For example, suppose each server has a 512MB memory-based file system cache. Now, if the total size of the frequently accessed content for the website served is 200MB, each server can cache the frequently accessed content separately; thus there would be no issue. However, if the total size of the frequently accessed content served is 1GB, each server cannot cache the frequently accessed content separately (since each server has only 512MB memory); this leads to cache misses on each server resulting in a loss of performance. On the other hand, if a number of servers (say four) form a network-based file system such as *PVFS*, the aggregate cache of the file system would be close to 2GB (512MB x 4 servers). Thus, all the frequently accessed content can be placed in the file system cache. This issue is especially of a great concern in shared data-centers. Since such data-center host multiple websites, the aggregate size of the *frequently accessed content* increases linearly with the number of websites.

As mentioned earlier, for dynamic content, the data used to produce the results (database) and the final output sent to the client are different. Thus, there are two kinds of caching that are possible: (i) caching the raw database objects themselves and (ii) caching the final output generated that is sent to the client. Caching the final output generated has several research issues associated with it, including maintaining consistency and coherency of the output file with the data objects present in the database, etc. We have done some previous work in this direction [18], but do not concentrate on that aspect in this paper. On the other hand, caching the raw database objects themselves is handled by the database itself.

In this paper, we study the impact of this issue for static (time invariant) workload. However, the ideas are also relevant for caching the raw database objects in dynamic content workloads.

3) Cache pollution effects: Caching has a significant impact on the performance of the web/proxy server. Each website in a data-center has a set of frequently accessed popular files. Due to the high frequency of accesses, these popular files tend to be highly sensitive to the caching capability of the file system. Ideally, we would like these popular documents to always be cached. While most of the requests in the website are for these frequently accessed files, some requests for other not-so-frequently-accessed files are also possible. Such requests tend to fetch the non-frequently accessed content to the file system cache, thus polluting the cache content and requiring later arriving requests for frequently accessed documents to result in a cache miss.

This issue is especially concerning for shared data-centers. With multiple websites sharing the same resources in a shared data-center, each website now has a lesser amount of file system cache to utilize. Further, since the frequency of access of different documents is different in a typical data-center environment (Zipf like distribution), the amount of degradation can increase in a super-linear manner as compared to the number of websites serviced in the shared data-center.

To understand the reason for this behavior in more detail, we classify the frequently accessed documents into two classes: ‘‘moderately hot” files and ‘‘very hot” files. As the names suggest, the most frequently accessed half of the popular files are classified as ‘‘very hot” files. Obviously, the number of requests for the ‘‘very hot” files is significantly larger as compared to the ‘‘moderately hot” files.

In a data-center hosting just one website, all the frequently accessed documents (both the ‘‘moderately hot” files as well as the ‘‘very hot” files) are present in cache. Now, accessing a large infrequently accessed document results in the frequently accessed documents to be evicted and results in some cache misses. Applying this in the shared data-center environment hosting two websites, since the amount of memory present on the data-center servers is constant, they can now cache only about half the number of popular files from each website; this would result in only the ‘‘very hot” files being cached from both the websites. Since these files are accessed a lot more frequently than the ‘‘moderately hot” files, they are significantly more sensitive to cache pollution effects. This results in a super-linear and sometimes drastic drop in the number of cache hits.

IV. Experimental Analysis

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 consider both local file systems (*ext3fs* and *ramfs*) as well as network-based file systems (*PVFS* and *Lustre*). In this section, we analyze the peak performance achieved by these file systems and study their impact on the client's response time and throughput in a multi-tier data-center environment. Especially, we study on network requirements, aggregated cache volume, and cache pollution effect for each file system. The analysis results show that the network-based file systems can provide a large aggregated cache volume while they have network requirements. We also observe that *ramfs* can sustain the cache pollution. To take the advantages of each file system, we also propose utilizing a multi file system in this section.

For all our experiments we used the following two clusters:

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.

Cluster2: A cluster system consisting of 8 nodes built around SuperMicro SUPER X5DL8-GG motherboards with ServerWorks GC LE chipsets which include 64-bit 133 MHz PCI-X interfaces. Each node has two Intel Xeon 3.0 GHz processors with a 512 kB L2 cache and a 533 MHz front side bus and 2 GB of main memory. We used the RedHat 9.0 Linux distribution.

Clusters 1 and 2 used the following interconnect:

Interconnect: InfiniBand network with Mellanox InfiniHost MT23108 DualPort 4x HCA adapter through an InfiniScale MT43132 twenty-four 4x Port completely non-blocking InfiniBand Switch. The Mellanox InfiniHost HCA SDK version is thca-x86-3.1-build-003. The adapter firmware version is fw-23108-rel-3_00_0001-rc4-build-001. The IPoIB driver for the InfiniBand adapters was provided by Voltaire Incorporation [14]. The version of the driver used was 2.0.5_10.

Cluster 1 was used as the server nodes in the data-center environment and Cluster 2 was used as the clients. We used Apache version 2.0.48, PVFS 1.6.2, Lustre 1.0.4, PHP 4.3.1 and MySQL 4.0.12 in all our experiments. Requests from the clients were generated using eight threads on each node.

A. Basic Performance of Different File Systems

In this section, we analyze the basic performance achieved by the different file systems: *ramfs*, *ext3fs*, *PVFS* and *Lustre*. We show the data transfer rates achieved by these file systems in two scenarios, transferring cached data and transferring uncached data. We also show the overhead in performing metadata operations such as file `open()`, `close()`, etc.

For all experiments, in the *PVFS* and the *Lustre* setups, three I/O nodes are configured with each I/O node using *ext3fs* as the local file system. The stripe size used was 64KB.

Table I shows three different measurements for the different file systems, (i) the overhead of metadata operations such as `read()` and `write()`, (ii) read latency for small (4K) and large (1M) cached content and (iii) read latency for small (4K) and large (1M) uncached content. Performance numbers for the `write()` operation and other file sizes are skipped here due to space restrictions and can be found in [24].

Comparing the metadata operations for the different file systems, since this is only a local operation for the local file systems (*ext3fs* and *ramfs*), the overhead can be expected to be low. On the other hand, since the network-based file systems (*PVFS* and *Lustre*) need to access their metadata managers to handle file permission issues, the overhead can be significant for them. Further, this implies that for small file transfers, the metadata operations can take up a significant portion of the transfer time in network-based file systems.

Coming to the data transfer rates, for files in cache, we can expect the performance of the local file systems to be better than the network-based file systems since its only a local operation. However for files not in cache, we see that the network-based file system give better or comparable performance compared to the local file system due to parallel accesses from different I/O servers. However it is to be noted that the numbers we reported here for read latencies do not include the file `open()` and `close()` overheads.

B. Network Requirements of file systems

In order to study the network requirements of different file systems in the data-center environment, we perform three sets of experiments. The first experiment evaluates the absolute amount of network traffic generated by each file system. The second experiment evaluates the rate at which data is requested over the network. These two experiments provide an indication about (i) the potential of the network becoming a bottleneck while using a network-based file system and (ii) the utilization of the network bandwidth provided by the data access patterns of the different workloads. The third experiment shows the end performance achieved by the different file systems under various workloads (Zipf Class0 through Zipf Class3) with a high temporal locality coefficient (α = 0.9). This experiment indicates the impact of the high overhead of metadata operations associated with network-based file systems.

Absolute Network Traffic Generated: Figure 4a shows the amount of network data transferred for a data-center environment hosting static content. We used the Linux utility *netstat* to monitor the network traffic and report the total number of packets received and sent from the web server. For the local file

TABLE I

FILE ACCESS LATENCY OF DIFFERENT FILE SYSTEMS

Latency (usecs)	ext3fs		ramfs		pvfs		lustre	
	4K	1M	4K	1M	4K	1M	4K	1M
Open & Close overhead	6	6	6	6	1060	1060	876	876
Read Latency (cache)	4	1602	4	1578	680	13825	7.7	1998
Read Latency (no cache)	1500	76312	1400	2379	9600	44108	3000	50713

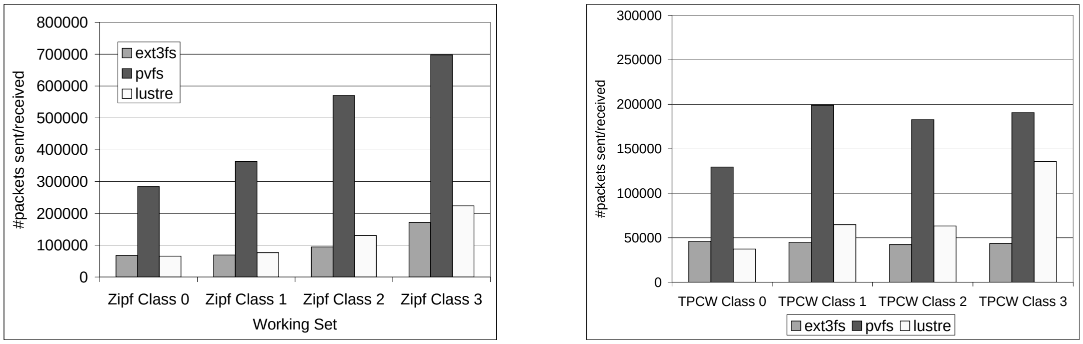

Fig. 4. Network requirements for various workloads: (a) Static Content (Zipf) (b) Dynamic Content (TPC-W)

system, the network requirements will be the total number of packets exchanged between the proxy and web server. However for the cluster file systems, in addition to the network traffic between the proxy and the web server, it would also include the communication between the web server and the file system servers for fetching the document over the network. We can see that the amount of network traffic associated with *PVFS* increases proportionally as compared to the local file system. For *Lustre*, however, the network traffic is very close to that of the local file system. This is due to client-side cache used by *Lustre*, i.e., *Lustre* maintains a cache of the fetched documents on the webserver nodes too thus reducing the requirement to go over the network for every request. The network traffic for *ramfs* is skipped in this graph and is expected to be similar to *ext3fs* since both file systems do not create any additional network traffic for file management.

Figure 4b shows the amount of network traffic generated for the different file systems for dynamic content (TPC-W browsing type of benchmark) in a data-center environment. As shown in the figure, surprisingly, the network traffic does not increase with the increasing workload size. For *ext3fs* the amount of network traffic almost remains constant mainly due the nature of these transaction-based queries. However even for the network-based file systems, the network traffic does not increase. The main reason for this is the indexing and caching capabilities of database systems. Database systems implement indexing and memory based caching as an effective means of query searching and processing. Several databases maintain this intelligent mapping and send file I/O requests for only the relevant non-cached data from the file system resulting in very less network traffic for network-based file systems. We see similar trend in *lustre* but the total number of packets received is similar to the *ext3fs*.

Data transfer rate: Figure 5a shows a snapshot of the I/O activity for static content workload in a data-center environment. We use the Linux *iostat* utility to monitor the I/O traffic at the web server tier. We see that *Zipf class 0*, *Zipf class 1*, *Zipf class 2* have very less I/O activity but for *Zipf class 3*, the amount of I/O read from the disk increases. For *Zipf class 4*, the amount of disk I/O activity is significantly larger compared to all workloads. For network-based file systems, this I/O activity would correspond to the network traffic generated. We observe that the rate at which data is accessed over the network is about 40MBytes/s. This is less than 25% of the network bandwidth provided by TCP/IP over the InfiniBand network we are using (TCP/IP achieves a peak bandwidth of about 200MBytes/sec in our testbed). In general, this rate of data transfer can be expected to be pretty low for most current networks.

On the other hand, as seen in Figure 5b, the amount of disk I/O for dynamic content is significantly lesser in comparison to the static workloads. This shows that the read and write patterns for databases are in forms of many bursts of small data reads or writes. Thus, using a network-based file systems for dynamic workloads might result in an under utilization of the network bandwidth.

TABLE II

WORKLOAD CLASSIFICATION

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

Overhead of Metadata operations and Network Traffic: To understand the overall impact of the high overheads of metadata operations in network-based file systems, we show the end performance achieved by the different file systems under various workloads (Zipf Class0 through Zipf Class3; described in Table II) with a high temporal locality (α = 0.9). Since the workloads have a high temporal locality coefficient and most files accessed are small files, the performance of these traces is dominated by the performance for small files.

Table III shows the performance of the data-center in Transactions per Second (TPS) for different single file traces (file sizes 4K and 1M) in two cases: when the data accessed is in cache and the case when the data accessed is not in cache, respectively. In both the cases, for small file sizes the network-based file systems perform worse than the local file systems due

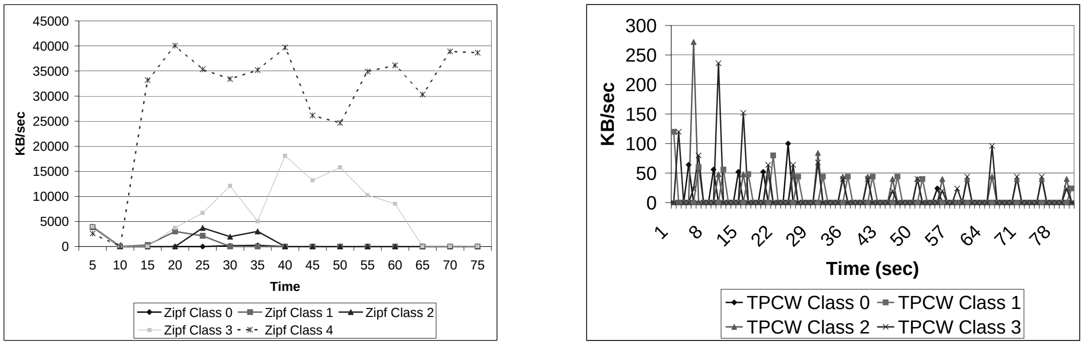

Fig. 5. I/O requirements for various workloads: Snapshot of the I/O activity (a) Static Content (Zipf) (b) Dynamic Content (TPC-W)

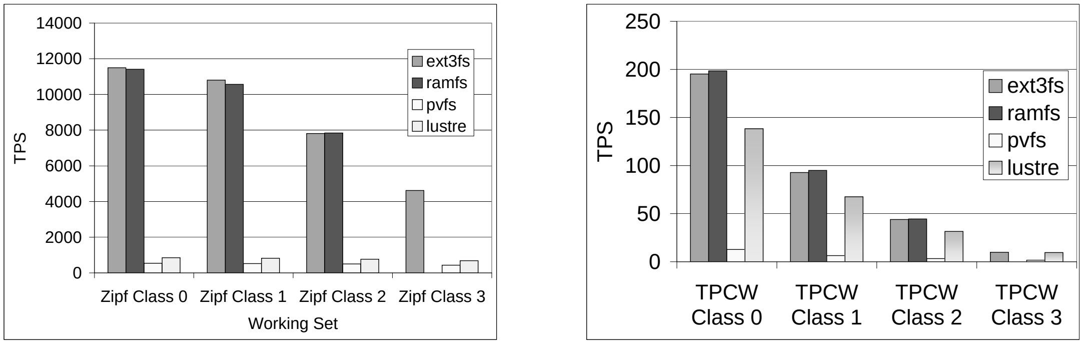

Fig. 6. Data-center performance with different file systems (a) Static Content (Zipf) (b) Dynamic Content (TPCW: Browsing Mix)

TABLE III

THROUGHPUT ACHIEVED BY DIFFERENT FILE SYSTEMS IN A DATA-CENTER

TPS	ext3fs		ramfs		pvfs		lustre	
	4K	1M	4K	1M	4K	1M	4K	1M
File in cache	11241	221	11378	253	699	211	1054	215
File not in cache	1488	66	2149	110	484	28.8	502	26.8

to the high file `open()` and `close()` overheads. For large files, network-based file systems perform either comparably or worse than local file systems. On the whole, for Zipf traces with a high temporal locality coefficient (which are dominated by small files), this overhead can be expected to be the worst.

Figure 6a shows the throughput achieved by various file systems on different workloads. With small workloads, since most of the files can be cached in the file system cache, *ext3fs* and *ramfs* achieve significantly better performance in comparison to *pvfs* and *lustre*. This degradation in the performance is mainly attributed to the high overheads of `open()` and `close()` operations for network-based file systems with small files. Further, since the workloads considered in this section show a high temporal locality, this would result in the most popular documents being fetched most of the time; these are small documents in our workloads, thus resulting in an overall degradation of performance.

Figure 6b shows the performance of data-center for different classes of dynamic content workloads (*T*PCW Class 0 refers to a workload with the TPC-W specifications, but the database size is equal to the working set size of the *Z*ipf Class 0 workload). We see that the data-center performance with *ramfs* or *ext3fs* is significantly better than the *pvfs* file system. One point to note is that most databases do not open or close the files corresponding to the database for every request. Thus, for the dynamic content workload, the network-based file systems do not have to face the overhead of file `open()` and `close()` operations. However, as shown in Figure 5b, the data accesses for dynamic content is in many bursts of small data transfers. This causes severe under-utilization of the network and can cause a drop in the performance for network-based file systems. This behavior is reflected in the performance of the *PVFS* file system compared to *ext3fs* and *ramfs*. *Lustre* on the other hand does comparably to *ext3fs* and *ramfs*. This is due to the client-side caching supported by it; the database server avoids the network traffic most of the times because of the cache maintained by the *Lustre* file system on the database node itself. We see similar trend for both shopping and browsing mix type of workloads and have been included in [24].

C. Aggregate Cache Size for file systems

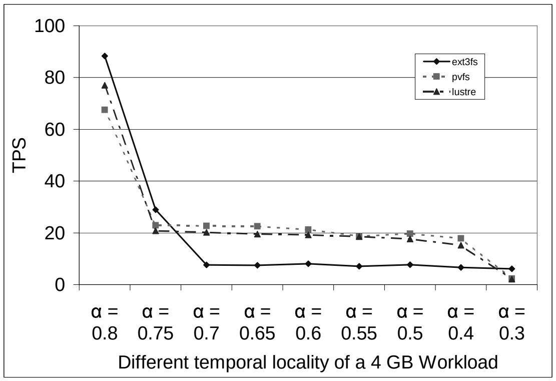

Fig. 7. Caching requirements for Static Workloads

To analyze the impact of the aggregate cache offered by network-based file systems, we measure the performance achieved by the data-center for various workloads. For all the workloads, we fix the working set size to 4GB and vary the α value to demonstrate workloads with different coefficients of temporal locality.

Figure 7 shows the transactions per second (TPS) achieved by a network-based file system (*pvfs*) and a local file system (*ext3fs*) for this type of workload. We see that, with an α value of 0.75, the data-center performance with *ext3fs* file system is better than *pvfs* due to the better cache hit time of the local file system. However, as the aggregate size of the popular content increases, in other words as the α value increases, the data-center performance with *ext3fs* is worse than data-center performance with *pvfs*. This is because of the larger aggregate cache size provided by *PVFS*. We see similar trend for the *lustre* file system as shown in Figure 7.

D. Cache Pollution Effects

In shared data-center environments, requests from multiple web-sites compete with each other to utilize the cache provided by the file system. Thus, requests for one website might pollute the file system cache causing the requests for the second website to be handled as cache misses.

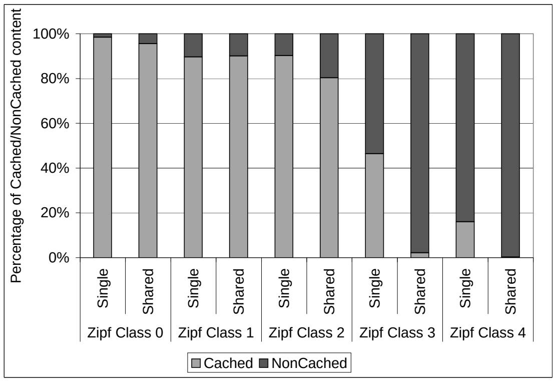

Fig. 8. Shared Data-Center Environment: Percentage of Cached and Non-Cached content for various workloads

In order to study this effect, we designed a test to study the amount of cache corruption that can occur for varying workloads. In this experiment, we compare the percentage of cache hits for different workloads in two scenarios. In the first scenario (legend *Single*), we host only one website on the data-center; this makes sure that most the “hot” files for that website remain in cache. In the second scenario (legend *Shared*), we host two identical websites on the data-center and compare the number of cache hits with that of the first scenario.

Figure 8 shows the total amount of cached and noncached accesses for the various workloads. This figure gives us several insights. First, for very small workloads (*Zipf Class 0* and *Zipf Class 1*) there is no difference in the number of cache hits. This is because the servers have enough memory space available to cache the frequently accessed documents for both the websites. Second, for medium ranged workloads (*Zipf Class 2*), the number of cache misses start increasing. This is because the servers do not have enough memory to cache all the frequently accessed documents for all the websites; thus increasing the number of websites hosted increases the cache misses.

Third, for large workloads (*Zipf Class 3* and *Zipf Class 4*), the number of cache hits drops drastically in the shared data-center scenario. Ideally, we expect the decrease in the number of cache hits to be lesser than 50% since the same cache is shared by two websites now; so about half of the popular documents from each website can still stay in cache. However, as the figure shows, this is not the case and the drop in the number of cache hits is by nearly a factor of *t*en. The reason for this is the impact of the requests for large non-frequently accessed documents. For a better understanding of this behavior, we repeat the example provided Section III-B.3 in the context of the results seen in this figure.

Again, we classify the frequently accessed documents into two classes: ‘‘moderately hot” files (lower half in the frequency of accesses) and ‘‘very hot” files (upper half in the frequency of accesses). In a single data-center, all the frequently accessed documents (both the ‘‘hot” files as well as the ‘‘very hot” files) are present in cache. Now, accessing a large infrequently accessed document results in some of the frequently accessed documents to be evicted and results in some cache misses. These evicted documents would most likely be the ‘‘moderately hot” files due to the lesser frequency of their access.

In the shared data-center environment, however, since the servers do not have enough memory space to accommodate all the popular files from both the websites, they can cache approximately half the popular files from each website; this would result in only the ‘‘very hot” files being cached from both the websites. Now, accessing a large infrequently accessed document results in some of the frequently accessed documents to be evicted. Since the file system cache does not contain any “moderately hot” files in this scenario, some of the ‘‘very hot” files will need to be evicted. Since these files are accessed a lot more frequently than the ‘‘moderately hot” files, they are significantly more sensitive to cache pollution effects. This results in a drastic drop in the number of cache hits for shared data-centers.

E. Multi File System Data-Centers

In the previous few sections, we have shown the impact of three broad issues in the shared data-center environment: (i) Network Traffic Requirements, (ii) Aggregate Cache Size and (iii) Cache Pollution Effects. Based on our observations on these broad issues, in this section, we propose utilizing a multi file system based data-center environment. This approach attempts to handle the above mentioned issues by utilizing each file system only for environments where it is most suited for, thus taking the best capabilities of all file systems.

The multi file system approach uses a combination of *ramfs* and *ext3fs* to serve websites serving (i) static content with a high temporal locality or (ii) dynamic content. Similarly the approach uses a combination of *ramfs* and either *P*VFS or *Lustre* for websites serving static content with low temporal locality. For simplicity, we will refer to *ext3fs*, *P*VFS and *L*ustre as the “backing file system” in the respective environments. In this approach, we place the most frequently accessed files in *ramfs* and the remaining files on the *b*acking file system.

We focus on three kinds of workloads in this section: (i) Static content with high temporal locality (Zipf Class 0 through Zipf Class 4), (ii) Static content with low temporal locality (Zipf

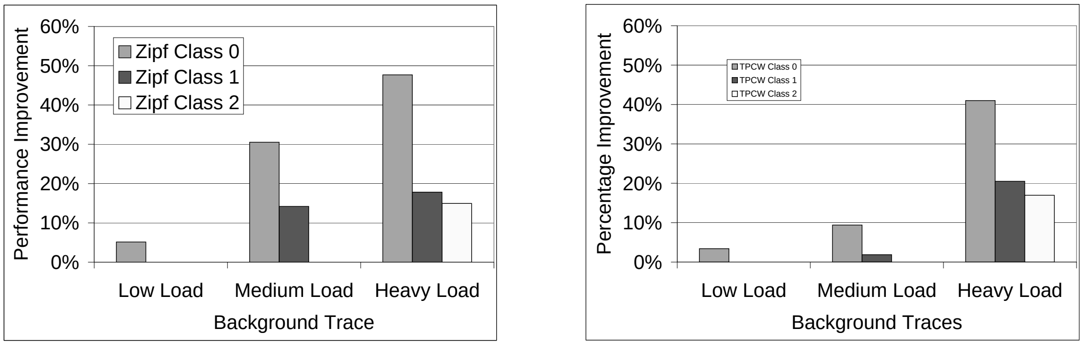

Fig. 9. Shared Data-Center Environment: Impact of a Hybrid file system (a) Static Content (Zipf) (b) Dynamic Content (TPC-W: Browsing Mix)

trace with varying α values) and (iii) Dynamic content

To show the impact in terms of performance, we emulate a web service provider hosting two web sites. For the first web site, we place all the files in the *b*acking file system. For the second web site, we set up two scenarios. In the first scenario, all files are placed in the *backing file system*. In the second scenario, all the "hot" files are placed on *ramfs*, and all other files in the *backing file system*. Requests for both web sites are serviced simultaneously. The performance difference of the second website between the two scenarios is our focus.

Figure 9a shows the percentage improvement achieved by the second web site when the websites serve static content with high temporal locality. As mentioned earlier, for this workload we use a combination of *ramfs* and *ext3fs* to store the files. We see that the multi file system approach achieves a performance improvement of up to 48% in some cases.

A similar strategy can be followed for dynamic content also. However, for dynamic content websites, the entire database is usually huge and may not be able to completely fit in the *ramfs*. Hence we may have to place only a part of the database in the *ramfs*. For example, we can place the tables that are frequently accessed in the *ramfs*. Though directly placing the database in a memory based file system might be inappropriate due to reliability and fault-tolerance issues, researchers have worked on semi-reliable databases and memory based fault-tolerant databases in the past. In this paper, we do not consider these issues and assume that the database is equipped with these features, thus allowing the usage of memory based file systems. In our experiment, we place the tables which are frequently accessed in *ramfs* and the remaining tables in the *backing file system*. For example in a TPC-W browsing type of benchmark, we realize that the 'select' queries are more frequently accessed than the 'update' queries. Also these 'select' queries predominantly access the *customer*, *item* tables frequently. Hence we place the files pertaining to these tables in *ramfs*.

Figure 9 shows the improvement achieved by the second web site while using *ramfs* to cache the frequently accessed tables which can fit in the cache. As mentioned before, 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 see a similar trend in comparison to static content; 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% with *TPCW Class 3* workload as the background trace. We see similar trend for TPC-W Shopping and Ordering type of benchmarks and also for real traces like RUBiS and RUBBoS. We have included those results in [24] due to space constraints.

Figure 10 shows the data-center performance for workloads with varying coefficients of temporal locality. We can see that placing the "hot" files in *ramfs* gives an improvement of up to 15% in some cases. There are two reasons for this. The first reason is the avoidance of cache pollution because of placing the "hot" files in *ramfs*. The second reason is the reduced overhead of the `open()` and `close()` system calls because of placing several small files in a local file system. As the temporal locality decreases, we see that there is a reduction in the amount of benefit achieved. This is because with very low temporal locality, the number of cache hits reduces; thus avoiding cache pollution cannot be expected to give high benefits. Secondly, with very low temporal locality, the number of requests for the smaller files reduces; thus the overhead of the `open()` and `close()` system calls is not as much of a concern.

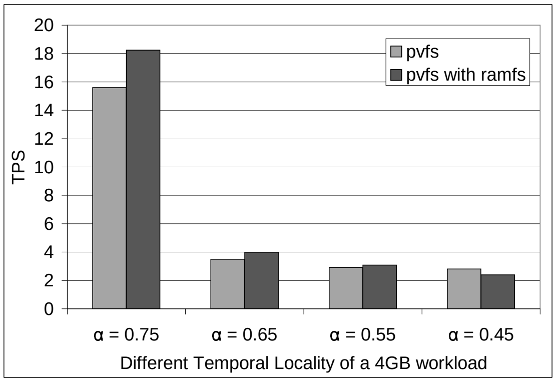

Fig. 10. Hybrid File System: Frequently accessed tables in *ramfs* and remaining tables in *pvfs*

V. Related Work

Hu et al. [13] have quantitatively analyzed performance bottlenecks of the Apache server and proposed several techniques to improve the same. Several others have studied the performance of a busy WWW server [1], [16]. Joubert et al. [15] proposed memory-based web servers in improving its performance. Most of the approaches mentioned above concentrate on evaluating only the WWW server whereas in our approach we evaluate a shared multi-tier data-center environment.

Wang et al. [25] have done detailed file system workload analysis. However the analysis was done for scientific applications and does not apply to data-center applications. Wang and Li [26] suggest a temporary file system which works in conjunction with the regular file system. However the web servers need to manage their own data and meta-data on raw disks and also the analysis was simulation-based. In the multi-file system approach we run real traces and identify the benefits of placing documents on ram disks in shared data-center scenarios. Martin F. Arlitt et al. [2] 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 al. [20] 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 propose a multi file system for multi-tier data-centers and identify its impact in shared data-center scenarios. We have also evaluated *PVFS* [27], [4] in our previous work and we expect the results to be similar over different high speed interconnects.

VI. Concluding Remarks

In this paper, we analyzed the impact of the file system in a shared data-center environment. We studied the impact of both local file systems (*ext3fs* and *ramfs*)and network-based file systems (*PVFS* and *Lustre*) in three broad aspects namely: (i) Network Traffic Requirements, (ii) Aggregate cache size and (iii) Cache pollution effects. We showed the capabilities and disadvantages of each file system in the light of the above mentioned three aspects. Finally, based on the insights gained from these broad issues, we proposed a multi file system data-center environment to utilize each file system only for environments where it is most suited for, thus taking the best capabilities of all the file systems. Our experimental results show that this approach can improve the performance by up to 48% in a shared data-center environment for static (time invariant) workloads showing high temporal locality, up to 15% for static workloads with low temporal locality and up to 40% for dynamic (time variant) workloads.

Dynamic reconfiguration of resources has been studied in the context of nodes [5], [3] and storage environments [19]. However, dynamic reconfigurability for caching and retrieving file documents based on support from the file system is quite novel and holds a lot of promise. The work in this paper was performed as an initial study to understand the implications of such dynamic reconfigurability with file system support. We plan to extend the knowledge gained in this study to implement a full-fledged dynamic reconfiguration module for file management.

VII. Acknowledgments

We would like to thank Dr. Jiesheng Wu for helping us with the PVFS component of this paper. We would also like to thank Weikuan Yu for his help in setting up the Lustre file system. Lastly, we would 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 insights into the implementation details.

References

[1] J. Almeida, V. Almeida, and D. Yates. Measuring the behavior of a world-wide web server. Technical Report 1996-025, 1996.

[2] M. F. Arlitt and C. L. Williamson. Web server workload characterization: The search for invariants. In *Measurement and Modeling of Computer Systems*, 1996.

[3] P. Balaji, S. Narravula, K. Vaidyanathan, H. W. Jin, and Dhabaleswar K. Panda. On the Provision of Prioritization and Soft QoS in Dynamically Reconfigurable Shared Data-Centers over InfiniBand. In *ISPASS*, 2005.

[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 *(ISPASS)*, 2004.

[5] P. Balaji, K. Vaidyanathan, S. Narravula, K. Savitha, H. W. Jin, and D. K. Panda. Exploiting Remote Memory Operations to Design Efficient Reconfiguration for Shared Data-Centers. In *(RAIT)*, 2004.

[6] RUBBoS: Bulletin Board Benchmark. http://jmob.objectweb.orgrubbos.html.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM (1)*, 1999.

[8] 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.

[9] A. Chandra, W. Gong, and P. Shenoy. Dynamic Resource Allocation for Shared Data Centers Using Online Measurements. In *Proceedings of ACM Sigmetrics, CA*, 2003.

[10] L. Cherkasova and S. R. Ponnekanti. Optimizing a content-aware load balancing strategy for shared Web hosting service. In *8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2000.

[11] A. Ching, A. Choudhary, W. Liao, R. Ross, and W. Gropp. Noncontiguous I/O through PVFS. In *Cluster Computing*, 2002.

[12] Cluster File System, Inc. Lustre: A Scalable, High Performance File System, 2004.

[13] Y. Hu, A. Nanda, and Q. Yang. Measurement, analysis and performance improvement of the apache web server, 1997.

[14] Voltaire Inc. http://www.voltaire.com/.

[15] P. Joubert, R. King, R. Neves, M. Russinovich, and J. Tracey. High-Performance Memory-Based web servers: Kernel and User-Space performance.

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

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

[18] 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.

[19] D. OHare, P. Tandon, H. Kalluri, and P. Mills. SNIA SSF Virtualization Demonstration. In *IBM Systems Group - TotalStorage Software: White Paper*, October.

[20] J. P. Patwardhan, A. R. Lebeck, and D. J. Sorin. Communication breakdown: Analyzing cpu usage in commericial web workloads. In *ISPASS*, 2004.

[21] H. V. Shah, D. B. Minturn, A. Foong, G. L. McAlpine, R. S. Madukkarumukumana, and G. J. Regnier. CSP: A Novel System Architecture for Scalable Internet and Communication Services. In *USENIX Symposium on Internet Technologies and Systems*, 2001.

[22] RUBiS: Rice University Bidding System. http://rubis.objectweb.org.

[23] 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.

[24] K. Vaidyanathan, P. Balaji, H. W. Jin, and D. K. Panda. Workload-driven Analysis of File Systems in Shared Multi-Tier Data-Centers over InfiniBand. Technical report, The Ohio State University, December 2004.

[25] F. Wang, Q. Xin, B. Hong, S. Brandt, E. Miller, D. Long, and T. McLarty. File system workload analysis for large scale scientific computing applications, 2004.

[26] J. Wang and D. Li. A light-weight, temporary file system for large-scale web servers.

[27] J. Wu, P. Wyckoff, and D. K. Panda. PVFS over InfiniBand: Design and Performance Evaluation. In *ICPP*, 2003.