A Comparative Experimental Study of Parallel File Systems for Large-Scale Data Processing

Zoe Sebepou[1], Kostas Magoutis, Manolis Marazakis, and Angelos Bilas[1]
Institute of Computer Science (ICS)
Foundation for Research and Technology - Hellas (FORTH)
P.O.Box 1385, Heraklion, GR-71110, Greece
{sebepou,magoutis,maraz,bilas}@ics.forth.gr

Abstract

Large-scale scientific and business applications require data processing of ever-increasing amounts of data, fueling a demand for scalable parallel file systems comprising hundreds to thousands of disks. Modern parallel file system architectures however, span a large and complex design space. As a result, IT architects are faced with a challenge when deciding on the most appropriate parallel file system for a specific scientific or industrial application in a large-scale computing installation. Typically, the right choice depends on the characteristics of the application as well as the design assumptions built into a parallel file system. In this study, we take a close look at two prominent modern parallel file systems, PVFS2 and Lustre, and compare them experimentally on a range of benchmark-driven scenarios modeling specific real-world applications.

1 Introduction

The growing need for processing ever-increasing amounts of data in large-scale scientific and business applications motivates research and development on parallel file systems that can offer scalable performance. Scientific simulations of various physical processes, data mining of large data sets to extract business intelligence, and enterprise-level operations such as e-mail services for large organizations, are examples of modern large-scale applications that require computing infrastructure comprising hundreds to thousands of processors. Such large data-processing requirements typically translate to high I/O throughput demands from parallel file systems comprising hundreds to thousands of disks.

When designing a large-scale computing installation for a scientific or industrial application, IT architects face a challenge deciding on the most appropriate parallel file system. The right choice typically depends on the specific characteristics of the application as well as the design assumptions built into the parallel file system. In this study, we address this challenge by taking a closer look at two prominent parallel file systems and comparing them experimentally on a range of benchmark-driven scenarios modeling specific real-world applications.

Our choice of parallel file systems to evaluate in this study, namely PVFS2 [12] and Lustre [5], is driven primarily by the increasing interest on these systems by the large-scale data processing community. Both systems share the fundamental design concepts of separation of data and metadata paths and the creation of I/O parallelism through direct client access to the storage servers [8]; however, they also differ in several other facets of their architecture, on issues such as consistency semantics, support for client caching, and metadata management. The experimental evaluation presented in this paper aims to highlight the areas where the systems perform comparably as well as the areas where their performance differs, using several industry-standard benchmarks.

A common problem with many benchmark-driven experimental evaluation studies is the difficulty of understanding the impact of a multitude of tunable parameters on the measured performance. A related problem is the difficulty of extrapolating the measured results to a wider range of applications. In this paper, we take a first step towards solving this problem by establishing a common framework based on a general model of applications performing parallel I/O. In this model, described in detail in Section 2, we capture typical application file-access patterns, attributes, and other characteristics that impact file performance. To understand the interaction of the application model with the fundamental design concepts behind PVFS2 and Lustre, in Section 3 we discuss their key similarities and differences, focusing on issues such as separation of control-data paths, parallelism, consistency, and access semantics. In Section 4 we describe

[1]Also with Department of Computer Science, University of Crete, P.O. Box 2208, GR-71409, Greece.

the benchmarks used in our study and in Section 5, the results of our experimental evaluation.

Our results show that:

- High bandwidth I/O for a large class of applications is achievable through the use of parallel I/O paths to file servers, offered by both PVFS2 and Lustre.
- Strict file-sharing semantics offered by Lustre are useful for certain applications but come at the expense of higher overhead for applications that do not require them.
- Efficient metadata management support in Lustre can be a differentiating factor for metadata-intensive applications requiring high transaction rates operating on large sets of small data objects.

Finally, in Section 6 we discuss related work, and in Section 7 we summarize our conclusions.

2 An I/O Model for Parallel Applications

Parallel applications typically access one or more files throughout their execution for the purposes of reading and writing their initial input, final output, or temporary data; taking snapshots of their state; saving statistics, etc. The I/O characteristics of a parallel application play a key role in the resulting performance, through their interaction with the underlying file system architecture, and can be characterized by a number of attributes, depicted in Figure 1.

It is important to note that the type of file access may vary for different phases of a parallel program. In general, a parallel program consists of a number of phases π_1 through π_n, each phase executed by a number of processes P_1 through P_m. During a phase π_j, a specific process P_i may be accessing a number of files f_1 through f_n, with each file f_k accessed either exclusively by P_i or concurrently with other processes. In this model, each triplet (P_i, π_j, f_k) is associated with a file access pattern characterized through the following attributes: (a) the type of the operations performed on f_k (e.g. read, write); (b) the average requested size in bytes; (c) the degree of sequentiality (e.g., sequential, strided, mixed, random); (c) the amount of I/O concurrency (e.g., number of outstanding I/Os per process). For files that are open for access by multiple processes, we distinguish between files in which the processes do not overlap, and files for which the processes do overlap. In the latter case, we specify the degree of sharing (i.e., number of bytes). Besides spatial overlap, we capture the degree of temporal overlap (i.e., simultaneous access to a file) by multiple processes.

Our intention in creating an I/O model of parallel applications is two-fold: First, we want to identify the different types of behavior underlying real-world applications, so that we can understand and quantify the demands they pose in the underlying parallel file system. Second, we want to capitalize on the predictive power of such a model by combining its instantiation for specific applications with experimental results obtained from standard or novel parallel file system benchmarks. Such benchmarks alone cannot address the question of "how does my application perform on parallel file system X?". A model-driven approach to application-specific performance prediction is similar in spirit to previous work on application-specific benchmarking [19] , which has also been applied to file systems [22].

Predicting application performance over parallel file systems, however, is a particularly challenging task, complicated by the multitude of system components involved (metadata and storage servers over storage controllers (RAID) over potentially thousands of disk devices) and the tiered architecture of most parallel file systems. File and storage allocation policies can vary significantly across parallel file systems and even across installations of the same parallel file system. As a result, reasoning about performance of an application requires taking into account specific characteristics of the underlying parallel file system. For example, whereas a specific phase of a parallel application appears to be accessing data in a (logically) regular manner, such as sequential or strided-sequential, the specific file allocation policy may in effect turn this to a near-random pattern towards the actual storage servers. Combining the aforementioned application model with such characteristics of parallel file systems in order to achieve accurate prediction of application performance is part of our ongoing and future work.

3 Overview of File System Characteristics

The parallel file systems used in this study, PVFS2 and Lustre, are targeted for large-scale parallel computers as well as commodity Linux clusters. A side-by-side comparison of the architectural concepts behind these systems, summarized in Table 1, reveals a number of similarities as well as a number of differences. In terms of similarities, both systems decouple metadata from data accesses, offering clients direct data paths to a shared object-based [8] storage pool addressable through a common namespace. One or more metadata servers are responsible for informing clients of the location of data in the storage servers but are not involved in the actual I/O operation. Both systems stripe data across the available storage servers, offering parallel I/O paths. Utilizing such parallel data paths, clients can achieve scalable I/O performance.

Key differences between the two systems are in their

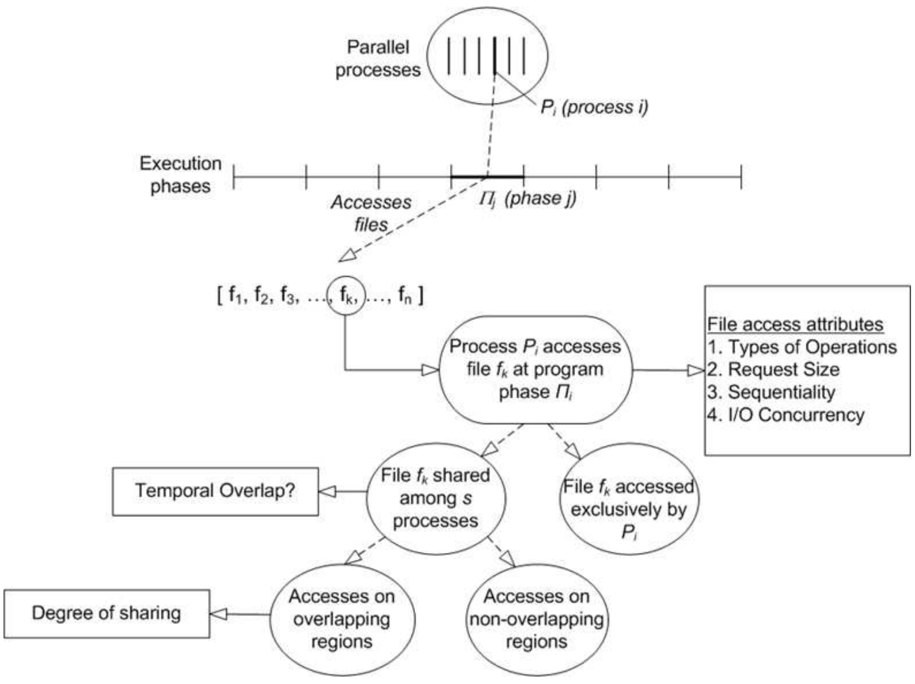

Parallel processes
P_i (process i)
Execution phases
Accesses files
Π_j (phase j)
[f_1, f_2, f_3, ..., f_k, ..., f_n]
Process P_i accesses file f_k at program phase Π_j
File access attributes
1. Types of Operations
2. Request Size
3. Sequentiality
4. I/O Concurrency
Temporal Overlap?
File f_k shared among s processes
File f_k accessed exclusively by P_i
Degree of sharing
Accesses on overlapping regions
Accesses on non-overlapping regions

Figure 1: I/O Model of Parallel Applications.

consistency semantics and in their caching and locking policies. Starting from file access semantics, PVFS2 offers the UNIX I/O and MPI-IO interfaces to applications. However, the UNIX I/O interface implementation in PVFS2 does not support POSIX atomicity semantics. Instead, it guarantees sequential consistency only when concurrent clients access non-overlapping regions of a file (otherwise referred to as "non-conflicting writes" semantics). On the other hand, Lustre's implementation of the UNIX I/O interface guarantees consistency in the presence of concurrent conflicting requests.

Support for POSIX consistency semantics in parallel file systems requires cache consistency across file system clients, typically implemented through the use of a locking mechanism. Lustre utilizes a scalable intent-based locking protocol whose functionality and management are distributed across file servers. With intent-based locking, a client that requests a lock on a directory in order to create a file in that directory, tags the lock request with information on the intended operation (*i.e.*, to create a file). If the lock request is granted, the metadata server uses the intention specified in the lock request to modify the directory, creating the requested file and returning a lock on the new file instead of the directory. In Lustre, each file server is responsible for the locking of the data stripes that it holds, avoiding a single lock manager bottleneck and increasing availability under failures. Lustre guarantees cache consistency by forcing the clients that have cached data to flush their caches before releasing any previously acquired locks.

Unlike Lustre, PVFS2 does not offer client-side caching or any file locking mechanisms. Lack of client caching fits well workloads with litttle or no data re-use. However, the application is responsible for I/O policies typically implemented in a file cache, such as prefetching. A benefit of not providing a client cache is the avoidance of the double-buffering effect when data is stored in both application and file system (client) buffers, increasing memory pressure [13]. Both file systems by default implement client write operations asynchronously (*i.e.*, without the need to synchronize with the underlying disk).

Lustre optimizes individual metadata operations by maintaining a cache of pre-allocated objects, thus decoupling allocation and assignment of metadata objects. Lustre further optimizes metadata operations by implementing a metadata-specific write-back (WB) cache. Through the use of that cache, a lock on a metadata object granted to a client allows the client to cache any modifications to that metadata object in its WB cache. PVFS's support for metadata optimizations includes a server-side attribute cache (disabled by default) and a client-side name cache (also disabled by default).

PVFS2 and Lustre differ in the way they distribute metadata management tasks for performance and availability. PVFS2 distributes metadata management responsibilities across any number of metadata servers in a partitioned, "share-nothing" manner that does not offer any

	PVFS2	LUSTRE
Striping Pattern	Configurable; defaults to round-robin	Round-robin
Stripe Size	Configurable; per directory	Configurable; per file/directory
Striping Width	Configurable; Up to # of available I/O servers	Configurable; Up to # of available I/O servers
Caching	No data caching in file clients; optional attribute caching	On file clients and file servers
Locking	Not supported by base protocol	Strong data, metadata locking at byte, page (4KB) granularity; Distributed intent-based locking
Semantics	"Non-conflicting writes"	POSIX semantics
Metadata Management	Multiple servers	Active-backup pair of servers

Table 1: Comparison of the PVFS2 and Lustre file system architectures. Both systems decouple data from metadata paths and achieve I/O parallelism through file striping.

Benchmark	**# Files**	**Sharing**	**Sequentiality**	**Overlap**	**Writes**	**Reads**	**Outstanding I/Os per client**
IOzone (r)	1	N/A	Sequential	N/A	0%	100%	1 or more
IOzone (w)	1	N/A	Sequential	N/A	100%	0%	1 or more
PIO write	*p*	N	Sequential	N/A	100%	0%	1
PIO read	1	Y	Sequential	Y	0%	100%	1
PIO read/write	1	Y	Sequential	Y	50%	50%	1
Cluster PostMark	*many*	N	Mixed	N/A	66%	33%	1
Tile I/O 1 (r)	1	Y	Strided	N	0%	100%	1
Tile I/O 1 (w)	1	Y	Strided	N	100%	0%	1
Tile I/O 2 (r)	1	Y	Strided	Y	0%	100%	1
Tile I/O 2 (w)	1	Y	Strided	Y	100%	0%	1

Table 2: Benchmark configurations used in this study: *p* stands for the number of client nodes (12 in our setup).

availability guarantees. Lustre on the other hand, can utilize two metadata servers in a primary-backup clustering scheme, thus achieving high availability, without however improving performance.

PVFS2 is implemented in both user-level and kernel versions whereas Lustre is implemented in the kernel. With regard to their metadata server architecture, PVFS2 uses the Berkeley DB database for mapping file handles to object references whereas Lustre uses a modified Linux ext3 in its metadata servers. Finally, PVFS2 includes support for increased parallelism in noncontiguous access patterns through an implementation of the *list I/O* interface [4].

4 Benchmarks

In this section we describe the benchmarks we use to evaluate the parallel file systems. The discussion and categorization of Table 2 follows the terminology of the application model described in the previous section. Several of the benchmarks use the MPI (Message Passing Interface [23]) programming model, which forms the basis for many parallel programs developed for the scientific domain. MPI provides the mechanisms to communicate between and synchronize processes executing on the processors of a computing cluster.

IOzone: In this highly configurable benchmark [1] a single process opens a file in order to perform a sequence of I/O operations with a certain profile (operation mix, access pattern, block size, etc.). IOzone is implemented as a single thread. However, multiple concurrent instances of IOzone can be explicitly started on a single client node.

Parallel I/O (PIO): This benchmark [21] produces a number of common spatial access patterns, such as strided, nested strided, random strided, sequential, tiled, and unstructured mesh. In this benchmark each process issues a number of I/O requests of a given size to either a single shared file or multiple exclusively-owned files. The total amount of data transferred (termed *buffer size*) is equal to the size of a single I/O request multiplied by the total number of I/O requests issued by each process.

Cluster PostMark: PostMark [10] is a synthetic benchmark aimed at measuring the system performance over a workload composed of many short-lived, relatively small files. Such a workload is typical of mail and netnews servers used by Internet Service Providers. PostMark workloads are characterized by a mix of metadata intensive operations. The benchmark begins by creating a pool of files with random sizes within a specified range. The number of files, as well as upper and lower bounds for file sizes, are configurable. After creating the files, a sequence of transactions is performed. These transactions are chosen randomly from a file creation or deletion operations, paired with a file read or write. A file creation operation creates and writes random text to a file. File deletion removes a random file from the active set. File read reads a random file in its entirety and file write appends a random amount of data to a randomly chosen file. In our evaluation we used a parallel version of this benchmark called Cluster PostMark, which was developed in our lab [7]. In this version, each PostMark process performs transactions within their own independent file set and synchronizes with all other processes over a global barrier at the beginning and end of the benchmark.

MPI Tile I/O (Tile IO): This benchmark [16] models scientific and visualization applications, which typically access file data structured in a two-dimensional set of tiles. In this benchmark, a number of processes concurrently access a shared file in which data has been laid out in a tiled pattern. A tile assigned to a specific process may or may not be overlapping with a tile assigned to a different process; we experiment with both cases and refer to the corresponding configurations as *Tile I/O 1* and *Tile I/O 2*, respectively. The access pattern is strided-sequential with a block size that depends on tile dimensions.

User-Perceived Response Time: Measuring the performance of interactive tasks is important as it attests to the quality of the day-to-day experience of human users. To estimate the response time of a typical interactive task as perceived by users of a parallel file system we evaluate the performance of the Unix `ls -lR` command on the Linux kernel tree (about 25,000 files).

5 Performance Evaluation

Our experimental setup consists of a 24-node cluster of Opteron x86 servers with 1GB of DRAM running Linux 2.6.12 and connected through a 1Gbps Ethernet switch. In this cluster we deployed Lustre v1.6.0.1 and PVFS2 v2.6.3. Half of the nodes are configured as file servers-one of them doubling as metadata server in both setups-and the rest as clients. Each node of each file system is provisioned with a dedicated logical volume comprising four 40GB partitions of SATA disks in a RAID-0 configuration. The total capacity of each parallel file system in the 12-server setup is about 1.7TB. Each file server node uses an underlying Linux file system of type ext3. In all MPI experiments we use the MPICH2 implementation [2].

To ensure that the benchmark workloads exceed cache capacity at clients and servers, thus producing significant disk I/O activity, we used a (shared) file of about 12GB (or about 1GB per client) in most cases. Files are laid out using the same file-system stripping policy, which is round-robin across all file servers using a 64KB stripe. While this is the default for PVFS2, we had to explicitly reduce Lustre's default stripe size of 1MB. However, we found that this change had minimal impact in our experimental results.

Client writes are performed asynchronously (*i.e.*, without flushing to disk) on both file systems in all benchmarks. Lustre's client cache uses a write-back policy, whereas PVFS2 does not perform client caching for data at all. Details on all benchmarks can be found in Section 4.

IOzone: In this benchmark we measure streaming performance reading and writing a large file stripped over all 12 file servers from a single client. The client process performs I/O with configurable block size (1KB, 64KB, 1MB, 4MB), waiting on completion of each I/O operation. We perform experiments using a single client thread (lstr-1thr, pvfs2-1thr) and four client threads (lstr-4thr, pvfs2-4thr).

Our results for the case of reads (Figure 2) indicate that both PVFS2 and Lustre can achieve the maximum achievable single-client bandwidth of about 110MB/s for large block sizes (upwards of 1MB) when using four client threads (lstr-4thr, pvfs2-4thr). For a 64KB block size and a single client thread, lstr-1thr reaches 90MB/s whereas pvfs2-1thr achieves only about 45MB/s. We believe that the difference is due to the lack of prefetching policies in the PVFS2 file client. As a result, performance under PVFS2 is constrained by the lack of I/O parallelism in the application. By increasing the number of simultaneously executing IOzone threads (pvfs2-4thr) we increase I/O parallelism

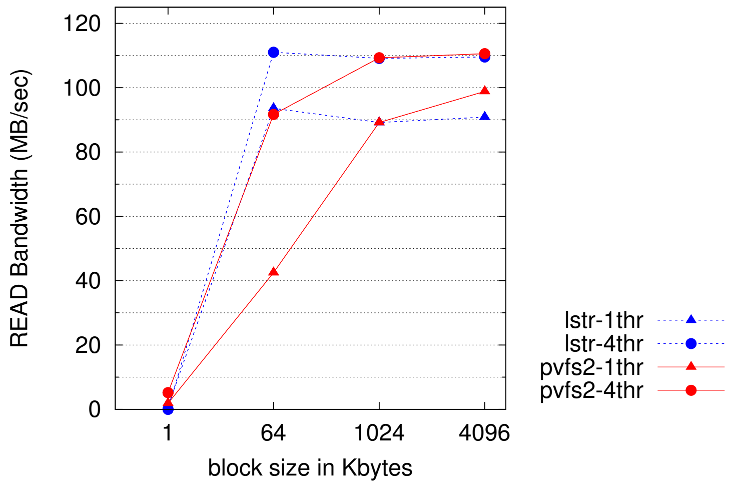


Figure 2: IOzone: read bandwidth; Single client, 12 servers.

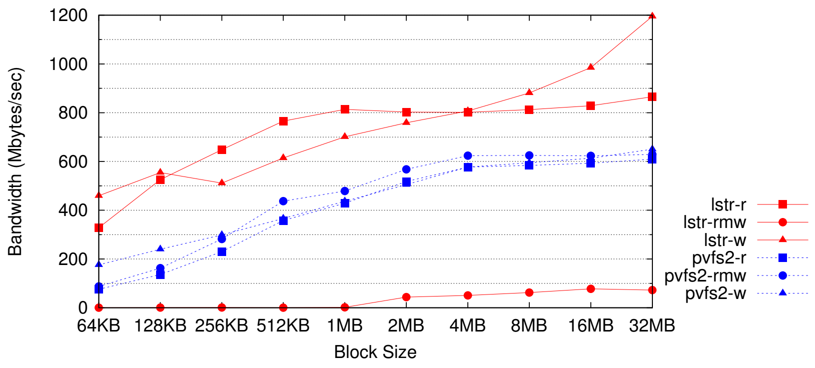


Figure 3: PIO: bandwidth; 12 clients, 12 servers.

at the client side, thus eliminating the performance difference between the two file systems. As can be seen from Figure 2, Lustre also stands to benefit from increasing application parallelism from one to four threads.

PIO: In this benchmark we measure sequential access by multiple file clients varying the block size from 64KB to 32MB, with each client process accessing a total of 1GB of data. For each block size we measure three configurations: (a) reads from a single shared file; (b) writes to separate, exclusively-owned files; (c) reads followed by writes (50%-50%) to a single shared file.

From the aggregate bandwidth results shown in Figure 3 we observe that PVFS2 performance scales consistently across the 64KB-4MB block size range across all configurations, leveling for larger block sizes (4MB-32MB). In contrast, Lustre performance varies across configurations. In the cases of reading from a shared file (lstr-r) or writing to separate exclusively-owned files (lstr-w), Lustre outperforms PVFS2 by at least 25-30% for all block sizes. For block sizes larger than 4MB, Lustre scalability improves to a nearly-linear rate when writing to separate exclusively-owned files. We attribute this difference to Lustre's use of a write-back client cache.

In the case of reads followed by writes to a single shared file (lstr-rmw), Lustre performance is hit by the overhead of maintaining strict consistency semantics across all 12 clients, reducing aggregate bandwidth to only a few MB/s for 64KB-1MB blocks, which increases to about 100MB/s for block sizes up to 32MB. This overhead is reduced in setups with fewer clients (and thus lower cache consistency maintenance cost) as we experimentally confirmed in a single-client single-server system.

To investigate the causes of the leveling in performance for PVFS2 (all configurations) and Lustre (reads; lstr-r) starting at 4MB blocks, we measured PIO performance with a single client and 12 servers, which we expected to be consistent with the IOzone benchmark for a single thread. We confirmed that expectation

	(a)	(b)	(c)
# files	100	800	8000
File size	10-100MB	1-10MB	4KB-1MB
Block size	64KB-2MB	16KB-1MB	4KB-128KB

Table 3: PostMark configuration parameters.

measuring about 90MB/s for both systems, which we believe is the asymptotic performance achievable by a single client when accessing a striped file over all 12 servers. Moving to 12 clients in the PIO benchmark, aggregate bandwidth for PVFS2 falls short of that achieved by Lustre by about 25%. We believe the reason lies in inefficiencies within PVFS2's metadata server involved in all I/O operations.

PostMark: We experimented with three configurations of Cluster PostMark with the following characteristics: (a) few large files: (b) a moderate number of medium-size files; (c) many small files. The parameters in these configurations are summarized in Table 3.

Looking at aggregate read throughput (depicted in Figure 4) for different block sizes, we observe that Lustre outperforms PVFS2 across all configurations and block sizes. In configuration (a), the benefit for Lustre is expected to be due to client caching, particularly in the absence of file sharing (*i.e.*, no contention for the cache), as well as its optimized metadata server architecture. As the benchmark becomes less I/O-intensive with decreasing file sizes, metadata management rises as the dominant source of overhead. In this case (configuration (c)), Lustre outperforms PVFS2 due to its more efficient metadata operations based on its support for intent-based locking [5] and pre-allocation of metadata objects. Our results are similar for the aggregate write-throughput (not shown).

Looking at aggregate transactions per second (depicted in Figure 5) for different block sizes, we observe that in the metadata-intensive case (configuration (c); large number of small files), Lustre outperforms PVFS2 by a factor of about 5 due to its more efficient metadata management operations. PVFS2 performance could in principle be improved by enabling additional (active) metadata server, a feature not supported by Lustre. In our experience however, we were not able to observe a measureable performance difference when using more than one metadata servers.

Tile-IO: In this MPI benchmark, a large file is logically partitioned into a two-dimensional grid of tiles. In this experiment we configure the benchmark for 12 tiles (6×2), each tile comprising $1254\times$ 1254 elements of 512 bytes each. In this logical structure, which is overlaid on a shared 9.5GB file, each file client is assigned and accesses a specific tile. We measure the cases of (a) non-overlapping file access (*i.e.*, clients do not access any common data); (b) overlapping file access in one dimension of the 6×2 grid. The block size with which MPI is performing file I/O is determined by tile dimensions and element size, in this case about 620KB.

This benchmark is expected to stress the file systems in more than one ways. First, access by each client to a dedicated tile (case (a)) requires effective strided-sequential I/O to non-contiguous regions in the file, resulting in a near-random access pattern (Section 2). Second, overlap in file accesses between clients in case (b) is expected to stress Lustre's cache consistency mechanisms. In contrast, PVFS2 is not expected to be impacted from sharing since it lacks support for file caching. In addition, PVFS2 avoids the double buffering effect between application buffers and file client cache, which is expected to impact Lustre.

Figure 6 reports aggregate read and write throughput respectively, in the cases of overlap and no overlap in tiles. Lustre outperforms PVFS2 by about 10% in aggregate read throughput (Figure 6 (a)) in both cases, aided by its more efficient metadata server implementation. Both systems benefit from overlap in tiles as they can take advantage of server-side file caching. In contrast, PVFS2 outperforms Lustre by a factor of two on aggregate write bandwidth (Figure 6 (b)) as Lustre is experiencing the overhead of maintaining cache consistency. Unfortunately, this overhead is not justified by the benchmark semantics which do not require strict consistency. Both systems benefit from write-back caching at file servers.

User-perceived Response Time: Interactive tasks common in day-to-day operations performed by human users are key to the adoption of parallel file systems by practitioners. To get a feeling for the performance of a typical such task, we measure the response time of a Unix `ls -lR` operation on all files in the Linux version 2.6.12 kernel tree (about 25,000 files). Our results are summarized in Table 4in the cases of a local file system (ext3), Lustre, and PVFS2. We observe that PVFS2 is about 40% slower than Lustre. Both parallel file systems, however, exhibit a significant slowdown (about an order of magnitude) compared to a local file system.

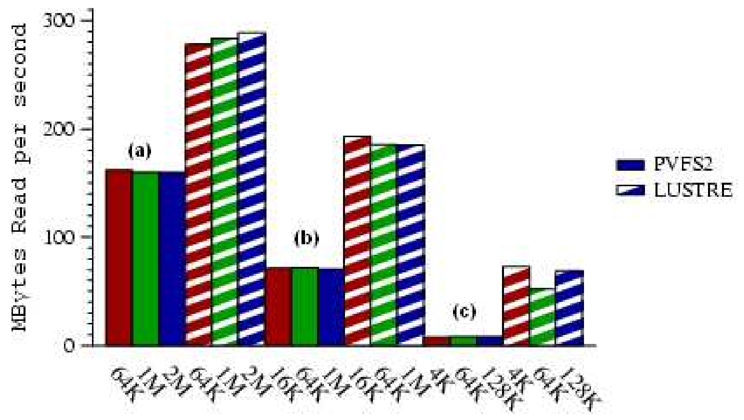


Figure 4: Postmark: Aggregate read throughput; 12 clients, 12 servers.

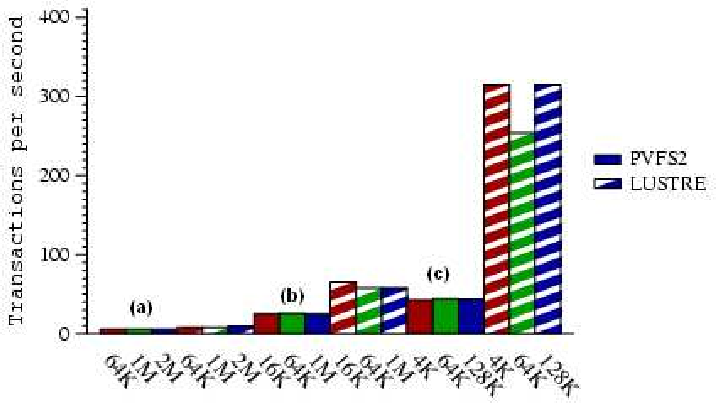
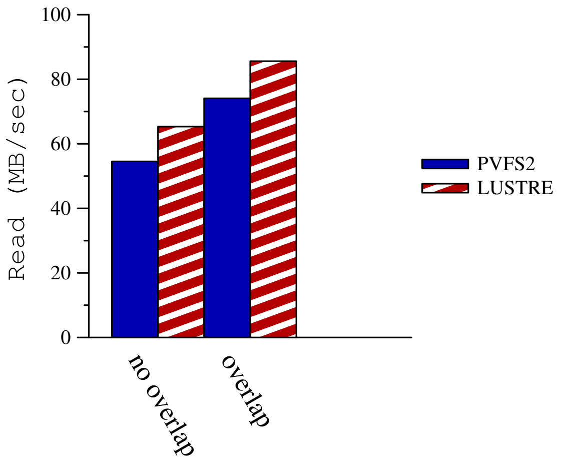


Figure 5: Postmark: Transactions per second; 12 clients, 12 servers.

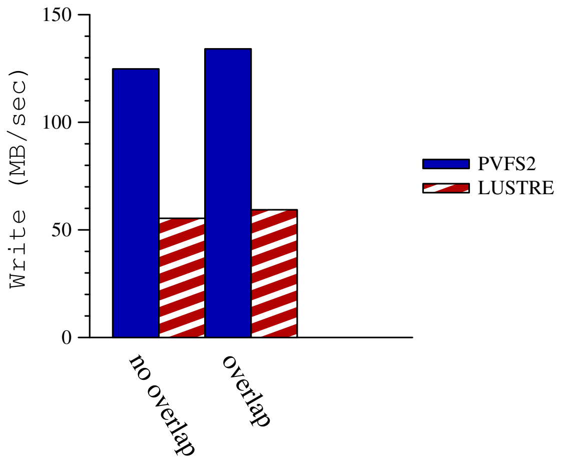
6 Related Work

The need of large-scale scientific and enterprise computing applications has motivated significant research in distributed file systems that can efficiently and reliably support processing large quantities of data. Distributed and parallel file systems, such as PVFS [12], Lustre [5], GPFS [20], GFS [18], AFS [9], and Panassas [24], have been developed to address the needs of a range of large-scale applications. A number of independent evaluation studies [3, 15, 14, 6, 11, 21] have explored various aspects of these parallel file systems.

An earlier experimental evaluation [3] at Lawrence Berkeley National Labs compared AFS, GFS, GPFS and PVFS to NFS [17] using PostMark, IOzone, and other benchmarks. A follow-on study [15] evaluated a number of shared-storage file systems on issues such as parallel I/O performance, metadata operations, and scalability using the MPTIO and METEBENCH parallel tests. Two related experimental evaluations [14, 6] of PVFS, Lustre and GPFS focused on I/O performance, scalability, redundancy, ease of installation and administration characteristics using the IOR parallel benchmark from Lawrence Livermore National Lab. Another study [6] focused on issues of installation, configuration, and management of these systems on two heterogeneous compute clusters using single-client and multiple-client bandwidth tests. This study concluded that all measured file systems outperformed an NFS installation and that PVFS2, Lustre and GPFS performed comparably across all their benchmark tests. Finally, Kunkel [11] focused on parallel file system performance bottlenecks using PVFS2 as a test case. This study introduced the idea of replacing PVFS2's methods accessing the underlying I/O subsystem with stubs diverting I/O from physical storage in order to avoid the performance impact of several sources of inefficiency in the I/O subsystem. Results provided for a wide range of contiguous requests as well as



(a) Read



(b) Write

Figure 6: Tile I/O: Aggregate read and write throughput; 12 clients, 12 servers.

File System	Response time (sec)
Linux ext3 (local)	5.5
Lustre	58
PVFS2	80

Table 4: User-perceived response time performing `ls -lR` on 25,000 files. Lustre and PVFS2 are configured as single client, 12 servers.

for metadata operations.

Our work relates to the aforementioned experimental evaluation studies in the focus on specific parallel file systems, including PFVS and Luster, but differs in that it provides a more comprehensive comparison spanning a wide range of benchmarks.

7 Conclusions

In this paper we performed a comparative experimental evaluation of two prominent modern parallel file systems, PVFS2 and Lustre, over a range of benchmarks modeling real-world applications. Our results indicate that both file systems are successful in offering parallel I/O paths to storage, achieving scalable performance over a broad range of applications. In areas where the file system architectures differ, such as in client caching, consistency guarantees, and metadata management, our results show that: (a) the cost of consistency management can be an unnecessary overhead for applica-tions that do not require it; (b) efficient metadata management can be a differentiating capability, critical for applications that require high transaction rates over large sets of small data objects. Beyond the specific benchmarks used in our experimental evaluation, we outlined a general model of parallel file access by applications, pointing to the key parameters that impact file access performance and as part of future work we expect to use this model to project our results and predict the performance of a broader class of applications.

8 Acknowledgments

We thankfully acknowledge the support of the European FP6-IST program through the UNIsIX project (MC EXT 509595), and the HiPEAC Network of Excellence (NoE 004408).

References

[1] IOzone Filesystem Benchmark. http://www.iozone.org.

[2] Mpich2. http://www.mcs.anl.gov/research/projects/mpich2.

[3] CHAN, S. Distributed File System Benchmarking. NERSC Lawrence Berkeley National Lab, http://www-pdsf.nersc.gov/ sychan/filesystems.html.

[4] CHING, A. Non-contiguous I/O through PVFS. In *Proc. of the IEEE Conference on Cluster Computing* (2002).

[5] CLUSTER FILE SYSTEM, INC. Lustre: A Scalable, High-Performance File System. White-paper, version 1.0.

[6] COPE, J., OBERG, M., TUFO, H., AND WOITASZEK, M. Shared Parallel File Systems in Heterogeneous Linux Multi-Cluster Environments. In *Proceedings of the 6th LCI Inter-*

national Conference on Linux Clusters: The HPC Revolution (2005).

[7] FLOURIS, M. Cluster PostMark. Personal Communication.

[8] GIBSON, G., NAGLE, D., AMIRI, K., BUTLER, J., CHANG, F., GOBIOFF, H., HARDIN, C., RIEDEL, E., ROCHBERG, D., AND ZELENKA, J. NASD: A Cost-Effective, High-Bandwidth Storage Architecture. In *Proc. of 8th Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (1998).

[9] HOWARD, J., KAZAR, M., MENEES, S., NICHOLS, D., SATYANARAYANAN, M., SIDEBOTHAM, R., AND WEST, M. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems 6*, 1 (February 1988), 55–81.

[10] KATCHER, J. Postmark: A New File System Benchmark. Tech. Rep. TR3022, Network Applicance Inc., 1997.

[11] KUNKEL, J.-M., AND LUDWIG, T. Performance Evaluation of the PVFS2 Architecture. In *Proc. 15th EuroMICRO International Conference on Parallel, Distributed and Network-Based Processing (PDP)* (2007).

[12] LIGON, W. I., AND ROSS, R. Overview of the Parallel Virtual File System. In *Proc. of Extreme Linux Workshop* (1999).

[13] MAGOUTIS, K., ADDETIA, S., FEDOROVA, A., SELTZER, M., CHASE, J., G. A., KISLEY, R., WICKREMESINGHE, R., AND GABBER, E. Structure and Performance of the Direct Access File System. In *Proc. 2002 USENIX Annual Technical Conference* (2002).

[14] MARGO, M. An Analysis of State-of-the-Art Parallel File System for Linux. In *Proc. of 5th LCI (Linux Clusters Institute) Conference* (2004).

[15] NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER, NERSC LAWRENCE BERKELEY NATIONAL LAB. http://www.nersc.gov/projects/GUPFS/results/filesystem/Filesystem.php.

[16] PARALLEL I/O BENCHMARKING CONSORTIUM. mpi-tile-io Benchmark. http://www-unix.mcs.anl.gov/pio-benchmark/.

[17] PAWLOWSKI, B., JUSZCZAK, C., STAUBACH, C., SMITH, C., LEBEL, C., AND HITZ, D. "NFS Version 3 Design and Implementation. In *Proc. of the USENIX Summer 1994 Technical Conference* (1994).

[18] RED HAT, I. GFS File System. http://www.redhat.com/gfs.

[19] SELTZER, M., KRINSKY, D., SMITH, K., AND ZHANG, X. The Case for Application-Specific Benchmarking. In *Proc. 7th Workshop on Hot Topics in Operating Systems (HotOS)* (1999).

[20] SHMUCK, F., AND HASKIN, R. GPFS: Shared Disk File System for Large Computing Clusters. In *Proc. of 2nd USENIX Conference on File and Storage Technologies (FAST)* (2002).

[21] SHORTER, F. Design and Analysis of a Performance Evaluation Standard for Parallel File. Master's thesis, Clemson University, 2003.

[22] SMITH, K. *Workload-Specific File System Benchmarks*. PhD thesis, Harvard University, 2001.

[23] SNIR, M., OTTO, S., HUSS-LEDERMAN, S., WALKER, D., AND DONGARRA, J. *MPI - The Complete Reference, Volume 1: The MPI-1 Core*. MIT Press, 1998.

[24] WELCH, B., UNANGST, M., ABBASI, Z., GIBSON, G., MUELLER, B., SMALL, J., ZELENKA, J., AND ZHOU, B. Scalable Performance of the Panasas Parallel File System. In *Proc. 6th USENIX Conference on File and Storage Technologies (FAST)* (2008).