Using Server-to-Server Communication in Parallel File Systems to Simplify Consistency and Improve Performance

Philip H. Carns
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
carns@mcs.anl.gov

Bradley W. Settlemyer and Walter B. Ligon, III
Dept. of Electrical and Computer Engineering
Clemson University
105 Riggs Hall
Clemson, SC 29634-0915
{bradles,walt}@clemson.edu

***Abstract*—The trend in parallel computing toward clusters running thousands of cooperating processes per application has led to an I/O bottleneck that has only gotten more severe as the CPU density of clusters has increased. Current parallel file systems provide large amounts of aggregate I/O bandwidth; however, they do not achieve the high degrees of metadata scalability required to manage files distributed across hundreds or thousands of storage nodes. In this paper we examine the use of collective communication between the storage servers to improve the scalability of file metadata operations. In particular, we apply server-to-server communication to simplify consistency checking and improve the performance of file creation, file removal, and file stat. Our results indicate that collective communication is an effective scheme for simplifying consistency checks and significantly improving the performance for several real metadata intensive workloads.**

I. Introduction

Due to the rapid increase in the number of multi-core, multi-processor nodes running applications in high-end clusters, the disparity between computational power and I/O throughput does not appear to be lessening. Parallel file systems are able to provide scalable I/O performance for applications accessing large contiguous file regions; however, the performance of metadata operations and smaller, unaligned file accesses still impinge upon performance for many types of scientific applications. A typical workflow for a large parallel application may require several data interaction steps [1], including:

1) Acquiring the data,
2) Staging/reorganizing the data onto a fast file system,
3) Analyzing the data,
4) Outputting results data,
5) Reorganizing the data for visualization,
6) And finally, processing the data for visualization.

Although high bandwidth data access is important for many of these steps, it is also critical to have efficient data management operations that reduce the time required to create files, remove scratch files, and query file permissions and properties.

As parallel file systems increase in scale, efficient metadata access becomes more difficult. Although client-driven serial metadata techniques may perform adequately for a few hundred clients accessing tens of metadata servers; when thousands of application processes attempt to create files, remove files, and list the contents of a directory, the performance of client-driven metadata operations directly impacts the number of storage nodes that can be deployed in a parallel file system. Additionally, the difficulty in maintaining a consistent view of the file system during independent and simultaneous multi-step metadata operations encourages file system developers to deploy complicated distributed locking approaches that increase fragility and further impact scalability. The use of server-to-server communication neatly addresses both of these problems. Server-to-server communication in a parallel file system improves the scalability of the file system by simplifying consistency control for metadata operations and improves performance by leveraging collective communication techniques to perform metadata operations more efficiently.

In the remainder of this section we describe related work in file system metadata management. In section 2 we describe the existing metadata algorithms in PVFS, a production-level parallel file system. We also propose alternative algorithms that leverage server-to-server communication in order to improve scalability and simplify consistency maintenance. In section 3 we describe our methodology for evaluating the performance improvements with both a simulator and a prototype implementation using PVFS. In section 4 we present our experimental results, and in section 5 we discuss our conclusions and possible further improvements available by extending the techniques described here.

A. Related Work

Devulapalli and Wyckoff measured the benefits of compound operations, handle leasing, and resource pre-creation strategies to improve file create performance in distributed file systems [2]. While all three optimizations are beneficial, resource pre-creation proved a particularly effective scheme for improving the performance of the file create operation.

The Lustre parallel file system uses intent-based locking to perform some types of metadata operations [3]. When

acquiring a lock to perform a metadata operation, the client sends along the operation request type. The server may then optionally choose to perform the operation directly rather than grant the lock. Multi-step metadata operations are still client driven; however, the locking overhead for multiple clients trying to perform similar operations on the same resource is reduced (e.g. multiple clients creating different files in a single directory).

GPFS uses a traditional distributed locking approach to maintain a consistent file view during file creation and removal [4]. To create a file, the client first acquires all of the locks necessary to create a new directory entry and a new inode. When all the locks are acquired, the client is then able to proceed with file creation. GPFS also uses a shared locking scheme that allows multiple clients to simultaneously update a file's metadata in a local cache. Before the cached data is committed to disk, all of the updates are serialized through a single file system client to ensure that updates are applied in a consistent manner.

The Ceph file system has explored mechanisms for distributing the metadata workload across multiple servers by dynamically partitioning the namespace, replicating metadata across servers, and committing metadata to storage in a lazy manner to limit metadata I/O [5]. Ceph's embedded inodes also lends itself to an efficient implementation of readdir_plus, an optimization that performs a file stat on an entire directory of files at once [6]. Their mechanism increases the amount of data retrieved per server interaction rather than using collective communication to improve the performance of the file stat operation for distributed files.

A parallel file system mirroring protocol based on server-to-server communication was evaluated using PVFS in [7]. The authors demonstrated that server-to-server communication was an effective means for simplifying consistency while replicating data across file system storage nodes.

II. Metadata Algorithms

The metadata algorithms presented here are taken from the Parallel Virtual File System (PVFS), an open source, production-level parallel file system. The PVFS file system runs as a set of distributed servers that act as a single file system (Figure 1). Servers may be configured to host metadata, file data, or both types of data. Client applications running on dedicated compute nodes access the file system through an MPI-IO driver, or using a Linux kernel module.

In PVFS a file is composed of a directory entry in the parent directory, a metadata object, and one or more data objects. The file system may be configured so that none of the file's parts are hosted on the same file system nodes, or one node may host all of the file's component parts. In this paper we have configured PVFS to use the most distributed approach, which has all of the servers configured to host metadata and data objects. This configuration allows the file system clients to have access to the largest possible aggregate bandwidth, although specific application workloads may favor other configurations. File metadata and directory entries are

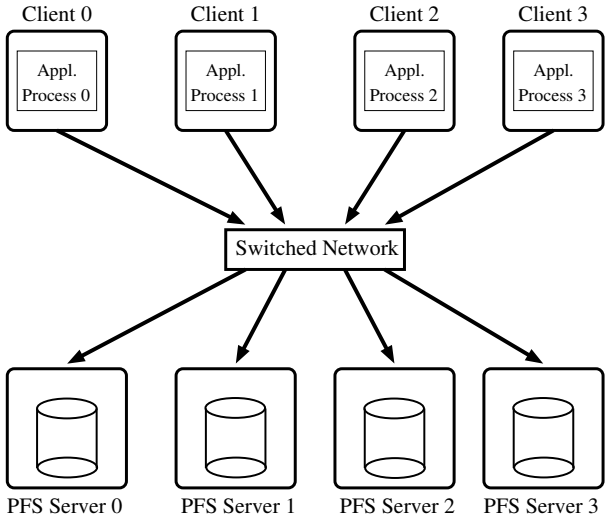

Fig. 1. PVFS Configuration

stored as a table of key-value pairs using the Berkeley DB libraries. The metadata object created for each file is assigned to a server for persistence in a round-robin order. File data is distributed across all of the data servers (one data object per server) and persisted in secondary storage using a file in the individual node's local file system. For these experiments we have not modified the PVFS metadata or data storage formats.

A. Collective Communication

The two major benefits of using collective communication for metadata operations in parallel file systems are simpler consistency semantics and improved performance. To perform the collective communication, we use a typical binary tree algorithm as shown in Figure 2. The collectives used are similar to the scatter and gather primitives; however, the servers do not each initiate the collective simultaneously. Instead, the nodes send *unexpected* messages to the recipients in the next level of the binary tree which then begin performing the necessary disk operations and simultaneously send data further down the binary tree. In our algorithms, the collective operations block until all of the participants successfully complete processing – the most popular semantic for file system metadata operations. If a participant cannot complete the collective operation (e.g. a local disk is full), the work previously completed in the collective will need to be undone. Error conditions are discussed more thoroughly during the description of the individual metadata operations.

The performance characteristics of collective communication are well described in literature [8]. The total communication cost of a serial implementation of the scatter and gather algorithms is described by the following equation:

$$T_{serial} = \alpha p + \beta n \quad (1)$$

Where p is the number of scatter recipients, α is the network start up cost, β is the network transmission time per bit, and n is the number of bits to be scattered. The communication cost

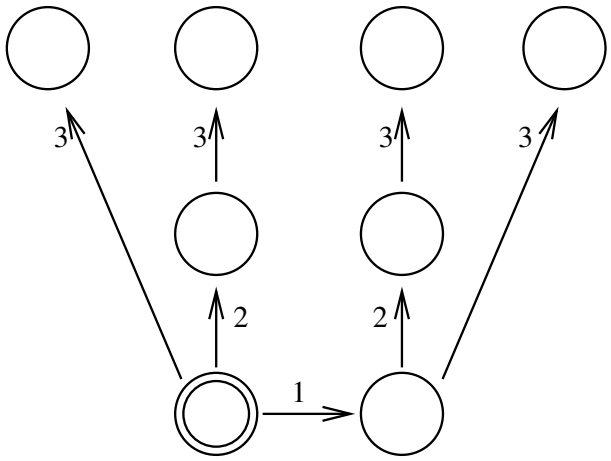

Fig. 2. Binary tree collective communication

of the collective scatter/gather algorithms is described by:

$$T_{coll} = \alpha log(p) + \beta n(p-1) \quad (2)$$

B. File Creation

The process of creating a file in a parallel file system requires a considerable amount of work. A metadata object must be initialized and populated with values, data objects must be initialized on each storage server, and a directory entry must be added in the parent directory. Although it is possible to perform some of the work in an "as needed" fashion; lazy creation techniques are unlikely to improve the performance of typical application work loads. Typical file creation use cases (e.g. copying files or data collection) immediately follow file creation with writing a substantial amount of file data to the file system. In such cases the notion that the file system can populate only the metadata on an eager basis, and then initialize the data storage resources on demand is unlikely to result in better performance. Object pre-creation strategies have been shown to improve performance [2], but our optimizations are independent of such techniques, and so we present metadata algorithms that do not rely on pre-created data objects.

The file creation algorithm listing (and all following listings) use the following conventions for brevity:

- C: represents a client process
- D: represents a data server
- M: represents a metadata server
- P: represents a parent directory server
- →: represents a request sent from a client to a server (response is implied)

For example, the statement "C → M create metadata object" indicates that a request was sent from the client to the metadata server in order to create a metadata object. Each algorithm step is implemented with atomic semantics. The client-driven file creation process is shown in Figure 3.

First, the client retrieves the parent directory's attributes to verify that creation permissions exists. The client then

```
1 C → P get parent directory attrs
2 C → M create metadata object
3 for each D:
4     C → D create data object
5 C → M set file attributes
6 C → P create directory entry
```

Fig. 3. Client-initiated file create algorithm

creates a metadata object on the metadata server, and creates the data objects on the data servers simultaneously. Once all the data objects are initialized, the file metadata is populated and the parent's directory entry is created. By ordering the operations carefully, the need for a distributed locking system that acquires all the resources preemptively is avoided. In the scenario where two clients attempt to simultaneously create the same file, only one client will be able to successfully write the directory entry; however, the client that fails to create the file will need to perform further work to clean up the orphaned metadata and data objects.

In addition to the possible leaked resources during creation failure, data object creation is slower than it needs to be. By making the client responsible for performing all the communication necessary to create the data objects, the server disk activity can be overlapped but the client's network link becomes a serialization point and a bottleneck for interacting with the data servers. The collective algorithm shown in Figure 4 resolves both of these issues.

```
1 C → P file create request
2 P locks parent directory
3 P → M create metadata object
4 for each D: (collective)
5     P → D create data object
6 P → M set file attributes
7 P creates directory entry
8 P unlocks parent directory
9 C ← P aggregate response
```

Fig. 4. Collective file create algorithm

In this algorithm, the server responsible for the parent directory entry is contacted by the client to perform the file creation. The parent directory server serializes local access to the parent directory and then creates the metadata object. The parent server and data servers collectively implement the binary tree communication algorithm shown in Figure 2 to create all of the data objects for the file. The parent server populates the metadata and creates the directory entry before unlocking the parent directory and signaling success to the client.

This algorithm simplifies consistency management because the parent directory can perform local serialization on the directory entry. There is no possibility of two clients making partial progress toward creating the same file. Additionally,

if the collective communication fails, the parent can simply unlock the parent directory locally rather than depend on a remote lock timeout mechanism. The algorithm also improves performance because the binary tree collective communication creates all of the data objects in time proportional to $O(\lg n)$ rather than $O(n)$ where n is the number of data objects for the file.

C. File Removal

File removal can be a very resource intensive operation for parallel file systems. File data may be distributed over tens or hundreds of data servers and orphaned data objects may result in a significant loss in storage capacity until the file system can be repaired (usually via an offline file system check). Figure 5 lists the basic client driven remove file algorithm.

```
1  C → M get file attributes
2  C → P remove directory entry
3  C → M remove metadata object
4  for each D:
5      C → D remove datafile object
```

Fig. 5. Client-initiated file remove algorithm

The client-initiated file removal algorithm demonstrates the difficulties in developing a file system server without the use of distributed locking. Consider the scenario where one client attempts to delete a file while another client is simultaneously modifying the file system permissions. Client 1 initiates the remove, and succeeds in removing the parent directory's entry for the file. Client 1 then attempts to continue the removal process by deleting the file metadata, but a network timeout occurs causing the metadata removal to fail. At the same time, client 2 modifies the permissions of the parent directory so that further modifications by client 1 are not allowed. Client 1 can then attempt to recreate the directory entry in the parent directory; however, since client 2 has modified the parent permissions, the file has been deleted. This outcome results in a large number of orphaned data files that waste significant storage space. Even in the case where the application code is simply interrupted immediately after the parent directory entry has been removed will result in orphaned data objects and wasted space until a file system check can be performed to recover the storage space.

The server-driven file removal listing in Figure 6 does not exhibit this behavior. By having the parent directory's server control the remove, access to the parent directory can be trivially serialized with a local lock, and a change in the permission's of the file targeted for removal is detected by the server and no repairs are necessary. Even if the client is interrupted, the parent directory server will fully complete the remove operation.

D. File Stat

File stat, while rare in parallel applications, is common during system administration and interactive data set management activities. One of the most common ways a user invokes

```
1  C → P aggregate remove request
2  P locks parent directory
3  P → M get metadata attributes
4  for each D and M: (collective)
5      P → D remove data object
6      P → M remove meta object
7  P remove directory entry
8  P unlock parent directory
9  C ← P aggregate response
```

Fig. 6. Collective file remove algorithm

the file stat command is using the UNIX utility *ls*. The *ls* command lists the contents of a directory and, optionally, each entries attributes (e.g. permissions, last modification time, file size). Efficient performance in the file stat command is critical to easing data management activities and improving the file system's interactivity. PVFS, like most file systems, employs a client-side attribute cache to avoid retrieving file metadata before every file interaction (e.g. to check permissions); however, metadata fields such as the file size and atime are not kept up to date for the same reason – to avoid writing metadata attributes after every successful file I/O. The file stat operation allows the user to query all of the file's metadata. The client-initiated file stat algorithm is shown in Figure 7.

```
1  C → M get metadata attributes
2  if file size is requested:
3      for each D:
4          C → D get data attributes
5      C compute logical file size
```

Fig. 7. Client-initiated file stat algorithm

One important feature to note is that the client is only required to contact the data servers if the user has requested the file size. The more common operation of requesting the file's permissions only requires checking the attribute cache and contacting the file's metadata server if the cache entry does not exist.

The collective file stat operation in Figure 8 differs from create and remove in that the file's metadata server initiates the collective communication rather than the parent directory server. Also, the collective data attribute request (Steps 5 and 6) acts like a gather operation instead of a reduction. The metadata server receives all of the data object attributes rather than just the computed size or latest access time. For simplicity, we prefer to perform the metadata calculations at the meta server, but a distributed reduction algorithm would be a reasonable improvement upon this algorithm.

III. Methods

In order to examine the runtime effects of the server-based collective communication operations proposed in this paper, we present both a simulation-based performance study and a

```
1 C → M aggregate stat request
2 M locks metadata object
3 M get metadata attributes
4 if file size is requested:
5     for each D: (collective)
6         M → D get data attributes
7     M computes logical file size
8 M unlocks metadata object
9 C ← M aggregate response
```

Fig. 8. Collective file stat algorithm

prototype implemented within PVFS running on two different platforms.

A. Simulation

We have developed a parallel file system simulator using the OMNeT++ simulation framework to more easily analyze and measure parallel file system extensions [9]. Our simulator uses the INET simulation models to provide an accurate representation of TCP/IP over an Ethernet network. We have implemented a detailed simulation model of a parallel file system and the underlying operating system using PVFS and Linux as our models. Our software is still under development and the disk and operating system models do not yet simulate all operating system and disk activity as accurately as we eventually plan to provide. However, in the case of small network-bound metadata operations our simulator was able to provide a representative model of how the modified version of PVFS performs with the proposed improvements. The simulator has been tuned at the individual message level to model the performance of the Adenine cluster at Clemson University. Our simulator is capable of modeling an arbitrary number of file system nodes; however, it currently only simulates a single connecting switch and so we have limited our scalability tests to a maximum of 500 compute nodes.

B. Prototype Implementation

In addition to a simulation study, we have also implemented the proposed optimizations in a widely available parallel file system, PVFS. Our experiments were performed using the number of file system servers as the independent variable and the operation execution time as the dependent variable. We gathered 35 samples for each file system configuration. The first two samples were discarded to avoid experimental noise related to startup costs. The remaining 33 samples achieved an approximate normal distribution about the sample mean and standard deviation. The presented data omits extreme outliers. An extreme outlier is defined as being less than $(Q_1 - 3(IQR))$ or greater than $(Q_3 + 3(IQR))$, where Q_1 and Q_3 represent the first and third quartile of the sample set, and IQR represents the maximum of the interquartile range or $5\mu s$.

C. System Configurations

1) Adenine: Clemson University's Adenine cluster is composed of 75 compute nodes. Each compute node contains dual Pentium III 1GHz processors with 1GB of RAM and 30GB Maxtor hard drives. All nodes are connected by a 100Mb/s (Fast Ethernet) network, while 48 nodes also have a 1Gb/s (GigE) network connection. Each network uses a single, independent dedicated switch. The compute nodes run a Linux 2.6 kernel with an ext2 file system.

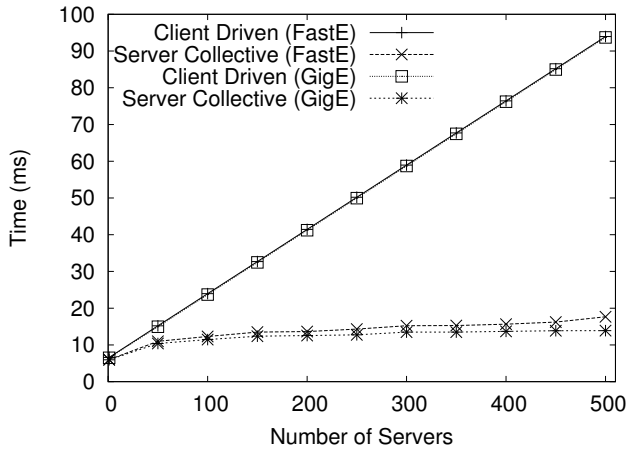

Fig. 9. File Create Performance on Adenine (Simulation)

2) Jazz: Argonne National Laboratory's Jazz cluster is made up of 350 compute nodes, each containing a 2.4GHz Pentium Xeon processor and at least 1GB of RAM. Each node contains an 82GB IBM hard disk. The compute nodes run a Linux 2.4.26 kernel with ext3 local file systems. The nodes are interconnected with both a 100Mb/s Fast Ethernet network and a high performance Myrinet-2000 network based on the PCI64C network interface card with a message latency of $6.7\mu s$ [10]. The transmission time of small messages sent during metadata operations are primarily limited by message latency, rather than the available bandwidth.

IV. Results

A. Simulated Performance Measurements

Figure 9 shows the simulated performance of the client driven and server collective file creation algorithms. The x-axis shows the number of file system storage servers and the y-axis shows the operation completion time in milliseconds. The performance results for both 100 Mbit/s (FastE) and 1 Gbit/s (GigE) interconnection networks are shown. Figures 10 and 11 show the relative performance for the two algorithms described for a single file remove and file stat.

In a parallel file system configured with 50 GigE storage servers, we note that the speedup achieved for file creation is 1.44. For file removal and file stat the speedups are 1.88 and 1.93, respectively. Figures 9, 10, and 11 clearly show the improvement from linear runtime to logarithmic runtime as predicted by equations 1 and 2.

Another, less obvious, benefit achieved in the collective metadata algorithms is the better utilization of the high bandwidth network. The simulated run times for the Fast Ethernet

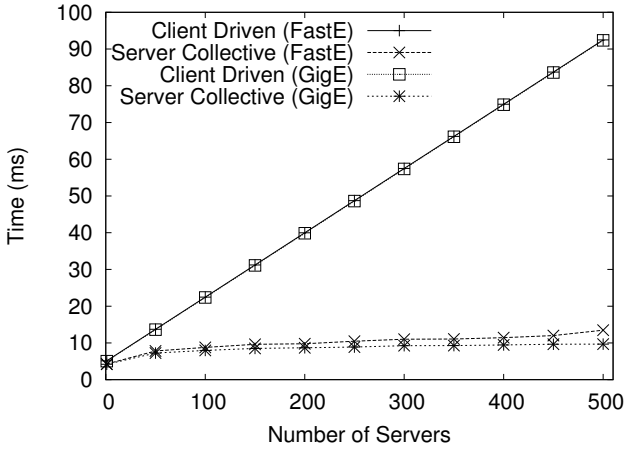

Fig. 10. File Remove Performance on Adenine (Simulation)

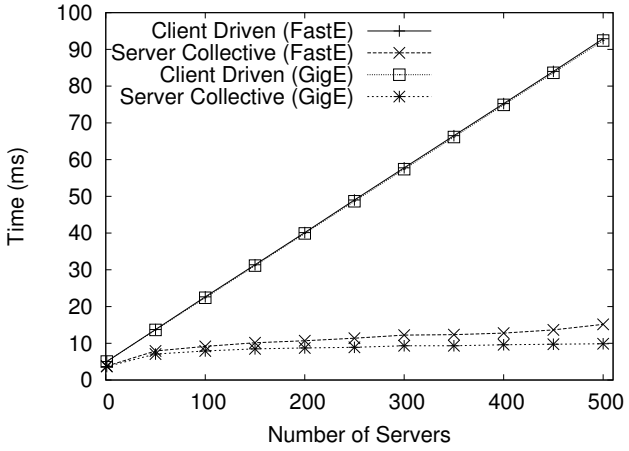

Fig. 11. File Stat Performance on Adenine (Simulation)

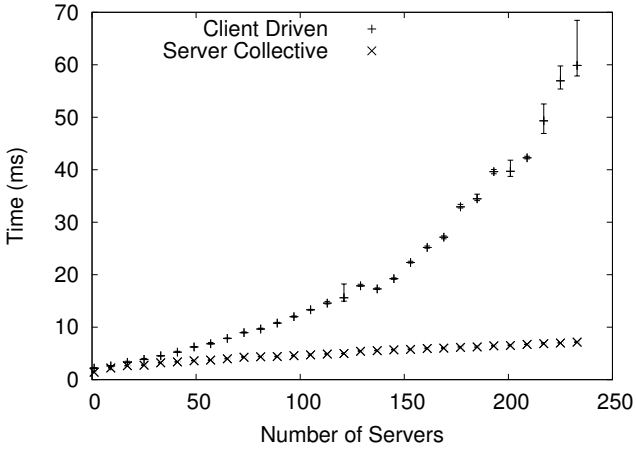

Fig. 12. File Create Performance on Jazz/Ethernet

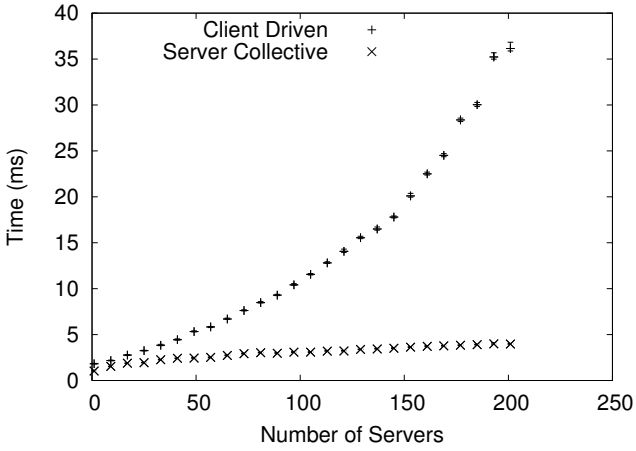

Fig. 13. File Remove Performance on Jazz/Ethernet

and GigE networks are nearly identical for the client driven algorithms because the individual messages are quite small and network performance is dominated by the setup cost (i.e. latency). The collective metadata algorithms require sending message sizes proportional to the depth of the tree. Thus, collective metadata algorithms will increase in performance (though, only logarithmically) as the bandwidth of the interconnect increases.

B. Prototype Performance Measurements

Figures 12, 13, and 14 show the measured performance of the client driven and server collective based metadata algorithms on the Jazz cluster using the 100Mbit/s Ethernet network. The range bars show the minimum and maximum observed execution times from each sample. For the server collective performance results the range bars are rendered within the individual line points, and cannot be easily distinguished. The slope of the client driven timing curves is increasing because the time spent in the client's poll system call increases proportionally to the number of servers [11]. The Linux kernel provided on the Jazz cluster does not provide the more modern epoll system call, which runs in constant time rather than in proportion to the number of open connections. We determined that a significant amount of the increasing overhead could be attributed to poll system call performance [12].

Additionally, as the number of servers increases, the probability of a long running interrupt occurring on one of the servers also increases [13]. The increasing range between the minimum and maximum recorded run times shown on the range bars demonstrates this phenomena. The simulation results shown in the last section do not exhibit the increasing slope because the server run times and overheads are deterministic rather than stochastic.

Figure 15 shows the prototype's performance results for a single file create on the Jazz cluster. The use of the Myrinet network significantly reduces the run time of file creations compared to the Ethernet results; however, the benefits of using collective communication are readily observable on configurations with as few as 20 storage nodes. When the file system is increased in scale to 60 storage nodes, the time spent

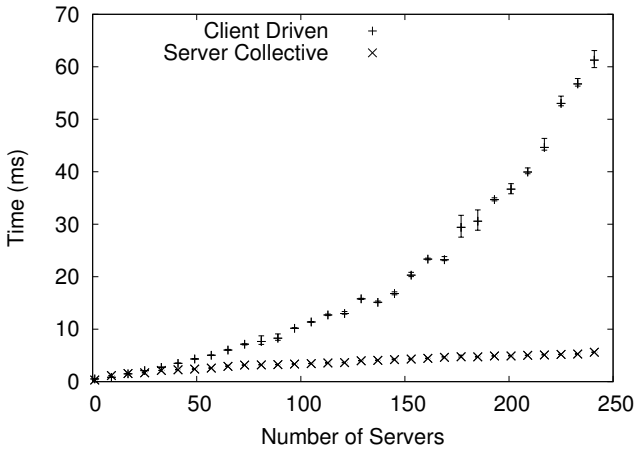

Fig. 14. File Stat Performance on Jazz/Ethernet

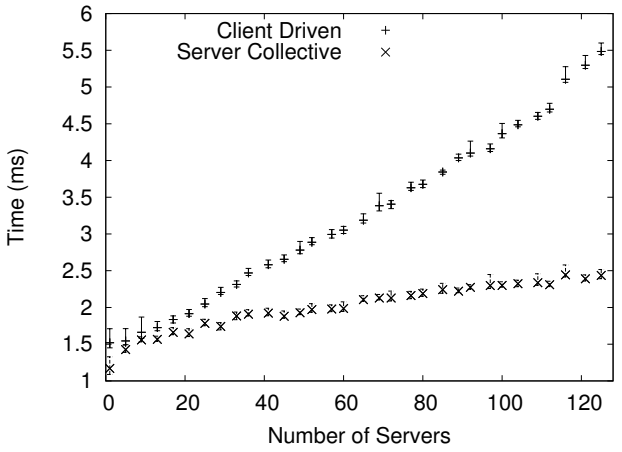

Fig. 15. File Create Performance on Jazz/Myrinet

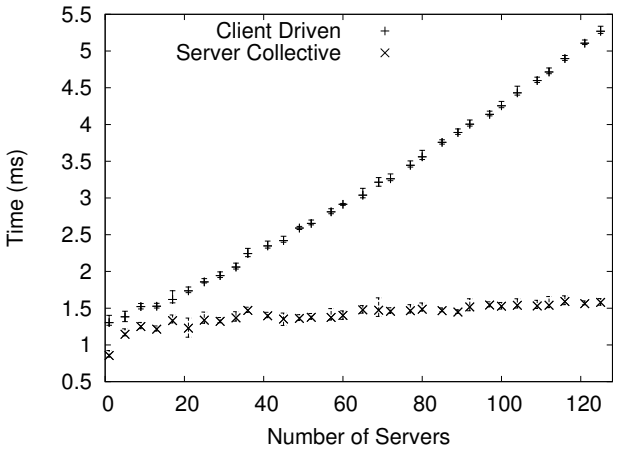

Fig. 16. File Remove Performance on Jazz/Myrinet

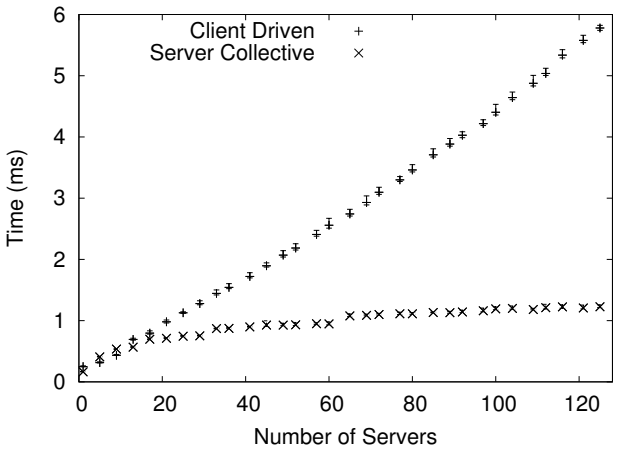

Fig. 17. File Stat Performance on Jazz/Myrinet

performing file creation with collective communication is only 65% of the time required for the client driven approach.

Figures 16 and 17 show the prototyped performance results for file remove and file stat using the Jazz cluster. The relative runtime performance is clearly changed from linear to logarithmic with respect to the number of file system servers. On a file system configured with 60 storage servers, the fraction of execution time required for file deletion and file stat with the server collective algorithms are 48% and 39%, respectively, of the client driven algorithms.

C. Interactive Workload Evaluation

We also measured the performance of the improved collectives in real world scenarios that require the manipulation of a large number of files. We chose three tasks commonly performed by both software developers and system administrators. In the first representative test, we extracted all of the files from a Linux-2.6.9 source file archive in tar format. In the second test, we performed a full listing of all of the files in the resulting source tree, which relied heavily on the implementation of the file stat operation. Finally, we performed a recursive remove of all of the source files. The 2.6.9 Linux kernel source is composed of over 1,000 directories and 16,000 files, with most of the files having a small or moderate size. Figure 18 shows the runtime in seconds for the three file manipulation tasks on a parallel file system configured with 74 dedicated storage nodes using both the simulation software and the prototype implementation running on the Adenine cluster (100Mbit/s network).

The measurements in Figure 18 exemplify the idea that optimizing the performance of the most critical operations can provide substantial execution time improvement. Each of the three tasks is composed of file system operations other than file creation, removal, and stat; for example, archived file extraction requires file I/O, directory creation, file creation, and metadata updates for atime. Still, the performance improvement of nearly 49% can be attributed entirely to the improved efficiency of the collective file creation technique. Similarly, recursive file listing and recursive file removal require additional metadata operations such as *readdir*; however,

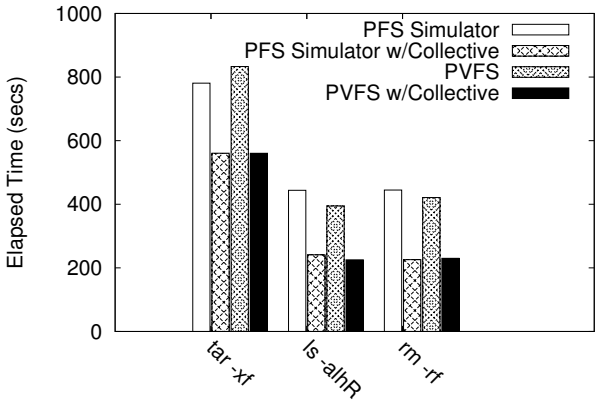

Fig. 18. Kernel Manipulation Performance on Adenine

the performance speedups in using collective file stat and collective file remove improve the benchmark execution times by 75% and 83%, respectively. In summary, with only the collective metadata operations presented in this paper, we were able to significantly improve the performance and interactivity of these three common developer and administrative tasks.

V. Discussion

We have described metadata algorithms that use server-based collective communication to simplify consistency and improve operation performance. Consistency is simplified because access to the metadata resource in contention is serialized on the server responsible for that resource. That server then uses server-to-server communication to drive the metadata operation to completion and signals success or failure to the client upon completion. The operation performance is improved because the server-to-server communication can be performed using collective patterns that parallelize network transactions and increase the scalability of parallel file systems. We have also shown that our performance improvements are not just applicable to slower commodity networks, but also provide significantly better scalability on low latency message passing interconnects. Finally, we have shown how our improved metadata algorithms impact three common file system tasks: file archive extraction, file listing, and file deletion. Although these tasks use many operations, by accelerating three fundamental metadata operations we were able to significantly improve the performance of the three tasks.

Our experiments included parallel file system configurations larger than most typical installations. In our simulator and prototype experiments we showed the performance improvements available on file systems using commodity networks to connect up to 500 storage nodes. We were also able to prototype the performance benefits of our modifications using 128 storage nodes connected with a high speed interconnect. In the future, we plan to continue exploring techniques to improve the scalability of parallel file system metadata operations and small file accesses. We also plan to continue improving our parallel file system simulator. We hope that our simulator will allow us to more easily evaluate candidate parallel file system designs and modifications for emerging cluster architectures.

In conclusion, we believe that techniques such as server-based communication in metadata operations are critical to increasing the scale of parallel file systems. The I/O bottleneck exists for many applications; however, in order to address the bottleneck, the scalability limitations of storage systems will need to be overcome as well. The scalability limitations do not exist simply as a performance problem to be overcome, but also as a resource control and complexity problem. Simplifying consistency control and improving performance will be critical to building large, high performance storage systems.

Acknowledgments

This work was supported in part by the National Science Foundation HECURA program under grant CCF-0621441. We would like to thank Dr. Robert Ross and the PVFS development team for their technical assistance in developing our prototype. We also gratefully acknowledge use of the "Jazz" cluster operated by the Mathematics and Computer Science Division at Argonne National Laboratory as part of its Laboratory Computing Resource Center.

References

[1] U.S. Department of Energy, "The office of science data-management challenge," March-May 2004.

[2] A. Devulapalli and P. Wyckoff, "File creation strategies in a distributed metadata file system," *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–10, 26-30 March 2007.

[3] P. J. Braam, "Scalable locking and recovery for network file systems," in *Petascale Data Storage Workshop SC07*, 2007.

[4] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proc. of the First Conference on File and Storage Technologies (FAST)*, Jan. 2002, pp. 231–244. [Online]. Available: citeseer.ist.psu.edu/schmuck02gpfs.html

[5] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller, "Dynamic metadata management for petabyte-scale file systems," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2004, p. 4.

[6] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320.

[7] B. W. Settlemyer and W. B. Ligon III, "A technique for lock-less mirroring in parallel file systems," in *Workshop On Resiliency in High-Performance Computing at 8th IEEE International Symposium on Cluster Computing and the Grid*, 2008.

[8] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing, Second Edition*. Pearson Addison Wesley, 1994, pp. 157–170.

[9] A. Varga, "The OMNeT++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference*, 2001.

[10] Myricom, Inc., "Myrinet GM 1.6.4 Performance," http://www.myri.com/scs/performance/Myrinet-2000/GM/gm-1.6.4-perf.html, April 2003.

[11] K. Shemyak and K. Vehmanen, "Scalability of TCP servers, handling persistent connections," *ICN '07. Sixth International Conference on Networking, 2007*, pp. 89–89, 22-28 April 2007.

[12] P. H. Carns, "Achieving scalability in parallel file systems," Ph.D. dissertation, Clemson University, Clemson, SC, May 2005.

[13] F. Petrini, D. Kerbyson, and S. Pakin, "The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q," *Supercomputing, 2003 ACM/IEEE Conference*, pp. 55–55, 15-21 Nov. 2003.