

Massive Data Processing on the Acxiom Cluster Testbed

Amy W. Apon, Pawel D. Wolinski, Dennis A. Reed, Greg M. Amerson, and Prathima Gorjala

Computer Science and Computer Engineering, University of Arkansas
{aapon,pwolinsk,dar01,gamerso,pgorjal}@uark.edu

ABSTRACT

The Acxiom Cluster Testbed is a high-performance, low-cost cluster that is being constructed at the University of Arkansas for the purpose of designing, constructing, and evaluating cluster architectures for massive data processing. Our evaluation efforts include performance testing of cluster hardware and a variety of middleware configurations for the cluster. One of the most important middleware components for massive data processing is a high-performance cluster file system. In this paper we present system requirements for a parallel cluster-based file system, our experimental evaluation of the NFS file system and the PVFS parallel file system for Linux, and future research goals.

Keywords: cluster computing, massive data processing, parallel file system evaluation

1. INTRODUCTION

One of the key difficulties currently faced by businesses is that of integrating information from a wide variety of sources to a single, usable form that can be used effectively in formulating a business or marketing strategy. This type of data integration typically requires searching and matching across very large heterogeneous data sources.

An example of the type of problem that might perform information integration over very large data sources is one faced by GE Capital Global Consumer Finance, UK. GE Capital is a global diversified financial services company and offers a variety of consumer services, including car leasing, home mortgages, and credit cards. GE Capital uses a massive data processing application to link together the different dimensions of its customer information to create a single profile of customers and households.¹ This profile enables the company to offer additional financial products and services that are tailored specifically to GE Capital retailers and customers. To perform the data integration required by GE Capital requires processing large-scale databases of up to hundreds of millions of customer records in real time.

High-performance applications such as these are often executed on expensive multiprocessor computers that use a shared-memory programming model. A typical example is a non-uniform memory architecture (NUMA) platform, consisting of 16 processors and 96GB main memory, and with a cost that ranges in the millions of dollars. The high cost for multiprocessor computing platforms has several obvious disadvantages. For example, it becomes very expensive to deploy multiple platforms to meet peak interactive demand. In addition, the dependence on a single large computer also leads to a dependence on the capabilities of the market and selected architecture. Performance difficulties may be observed with applications that do not fit into memory and require frequent disk access. As a result of these cost and performance factors, cluster computing is an attractive alternative for massive data processing applications.

One key component of a cluster architecture for massive data processing applications is a high-performance file system that can serve data to the compute nodes with very high bandwidth and high reliability. In the remainder of this paper we present requirements for a parallel cluster-based file system, and we present our evaluation of NFS and PVFS, a Linux-based parallel file system, as possible solutions for the data and file needs of the applications. We also describe our future goals for cluster-based file system research.

This work has been supported by grant #ESS-996143 from the National Science Foundation, by a grant from Acxiom Corporation, and by a matching grant from the Arkansas Science and Technology Authority.

2. PARALLEL FILE SYSTEM REQUIREMENTS

Parallel file systems is an active area of research, and several recent projects address parallel file access in a cluster architecture.²⁻⁶ Despite active research in parallel file systems the NFS central-server file system is a commonly used file system for many low-cost clusters. NFS version 2 has been available since 1985, and NFS version 3 was the first to offer support for large files, among other improvements.⁷ NFS offers a client-side cache, weak cache consistency, and a stateless server. The NFS protocol does not specify how the server is to implement the interface, or how a client should manage cached data.

We are considering several strategies for parallel file access, including building our own parallel file system. Several design and implementation requirements exist for a parallel file system to be useful for massive data processing. These include:

1. The system should be Linux-based. Linux system offer high performance for very low cost. Many cluster middleware and supporting tools are already available for Linux. The file system should also run on and be optimized for Linux. Optimizations that may be desirable include the ability of the file system to use native Linux buffer caching facilities, and application optimizations that can work in combination with the virtual memory tools available in Linux.
2. The application should not have to be recompiled to use the file system. Also, we make no assumptions about the language or tools used in the implementation of the application. While our performance testing described in this paper uses Message Passing Interface (MPI)⁸ as a tool for spawning processes and running the experiments, we do not want the file system to be dependent on MPI or any other specific middleware tool.
3. The file system should run over commodity networks, which implies the use of TCP/IP for communication. It is possible that the application and file system may be distributed over a wider geographic area than just a single lab room. However, we would like the application to use low-latency messaging where it is available.

The Parallel Virtual File System (PVFS) is a parallel file system for Linux.² The design goals of PVFS include high bandwidth data transfer to and from multiple processes in the cluster, the ability of the application to access the file system without recompilation, support for common Linux shell commands related to files, such as `ls` or `cp`, and ease of installation and use. While other file systems are also available, PVFS was chosen for benchmarking because of its availability and general ease of use. The benchmark information gained from the use of PVFS will help us to determine our future direction in file system research.

PVFS is designed as a client/server system, with server daemons running on I/O nodes to receive and process requests. PVFS uses TCP/IP for communication. It can be installed as a module and mounted using the Linux mount command. PVFS stripes files using uniform-sized stripes across the I/O nodes in the cluster. The metadata information about the file includes the list of I/O nodes that contain file data and the stripe size. An I/O node may also be a compute node in the system.

The metadata for all files in the PVFS system is maintained using a single manager daemon. When a file is opened, the client process contacts the manager daemon to retrieve the metadata information. The metadata information is in the form of {base I/O node number, number of I/O nodes, stripe size}, and fully specifies the location of data in the file. Reading from a file then only requires a request and response from each I/O node that contains stripes of the file request. The I/O manager is contacted for file operations such as open, close, and delete but is not contacted for read or write operations. Also at file open, the client establishes TCP/IP connections to the I/O daemons.

An ordered set of I/O daemons runs on the I/O nodes, one per node. Clients communicate with the I/O daemons directly to request data. The request is in the form of a descriptor of the file region being accessed. The I/O daemon determines locally what portion of the file it owns and then performs the necessary data transfers. The simple mapping allows for no additional control messages to be needed for reads and writes at the application layer. The local file system is used for storing the file stripes. The local file name is based on the inode number that the manager assigned to the file.

3. PERFORMANCE EVALUATION RESULTS

A simple performance study was designed to test how a parallel file system such as PVFS might improve the application performance of applications that perform massive data processing as compared to the use of NFS. We found that most performance testing of NFS in the literature focuses on the performance of the cache for read operations, or the performance of write operations.^{7,9,10} However, for massive data processing applications, a significant portion of the workload is reading all or portions of very large data files, either sequentially as for a load operation, or one time, as for a match operation. For these types of workloads the cache may not improve the performance of the application. The performance tests in this paper focus on workloads that involve reading very large files.

The new Acxiom Cluster Testbed hardware was not available at the time of the experimental study, so the experimental platform for the results in this paper is our existing cluster at the University of Arkansas which consists of four Compaq Pentium III 500MHz computers and four Gateway Pentium II 450MHz computers. The eight computers are connected via an HP ProCurve 100Mbps switch. Each Compaq computer is equipped with 256MB memory and an EtherExpress Pro 10/100 Ethernet card. Each Gateway computer has 128MB memory and a 3Com 10/100 Ethernet card.

The experimental cluster is configured with a variety of hard drives. The theoretical throughput of each hard drive was measured by the Linux utility `hdparm`. Node 0 of the cluster is configured with two IDE hard drives. One of the hard drives has a capacity of 40GB and holds only a copy of the experimental file in an NFS-mounted directory. The throughput on this drive was measured to be 19.28MBps. The second hard drive in Node 0 holds other system files and its portion of the PVFS-mounted test file. The second hard drive has a measured throughput of 12.65MBps. The hard drives in the remaining nodes hold only their portion of the PVFS-mounted test file. Three of the cluster nodes (Nodes 1, 2, and 3) are equipped with IDE hard drives that have a measured throughput of around 13MBps. Three of the cluster nodes (Nodes 4, 5, and 6) are equipped with SCSI hard drives that have a measured throughput of around 18MBps. The remaining cluster node (Node 7) is equipped with an IDE hard drive with a measured throughput of 18.44MBps. The throughput of all drives exceeds the theoretical throughput of 12.5MBps of the Fast Ethernet network that interconnects the nodes.

System software on the cluster is RedHat Linux version 7.1 with kernel version 2.4.2. MPICH version 1.2.1 of MPI was used to spawn processes on the cluster nodes and to synchronize the experiments. NFS protocol version 3 was used, and PVFS version 1.5.1 with PVFS kernel version 0.9.1 was used for the parallel file system.

A test file with a fixed record format of 838 bytes, approximately 30GB in length, was constructed for the testing. A shortened version of the file was used to make the experiment timings reasonable and to allow the file to fit on the available disk space in our existing cluster. For each experiment a 1GB file, which is larger than the aggregate main memory in the cluster, is read from disk. The performance reported in all experiments is throughput in MBps.

For the NFS portions of the experiments the test file was located in an NFS-mounted directory on the fastest hard drive on Node 0 of the cluster. In order to model a cluster architecture that is independent of a front-end application server, the NFS experiments do not utilize the NFS server as one of the participating nodes that access the file. Because of that, up to seven nodes participate in reading the file during the NFS experiments.

For the PVFS portions of the experiments the test file was striped equally across all eight nodes of the cluster. The stripe size is a configurable option in PVFS. The stripe size used is 8KB, distributed in round-robin fashion across nodes 0 up to 7 repeatedly.

Both NFS and PVFS are designed to take advantage of a server-side cache through the Linux buffer cache system. Efforts were made to ensure that the server-side cache was empty at the start of every file experiment. In the case of NFS the file was unmounted and remounted, which caused the system to clear the server-side cache. In the case of PVFS unmounting was difficult because of the many remote daemon processes that had to be stopped and restarted. For PVFS, a program was executed that loads a file into memory on each node that is the size of that node's local memory to ensure that the cache was emptied of the test file.

Experiments were designed to be similar to operations that might occur with a massive data processing application and are typical of other file system studies in the literature.¹¹ Three kinds of file read experiments were performed, as follows:

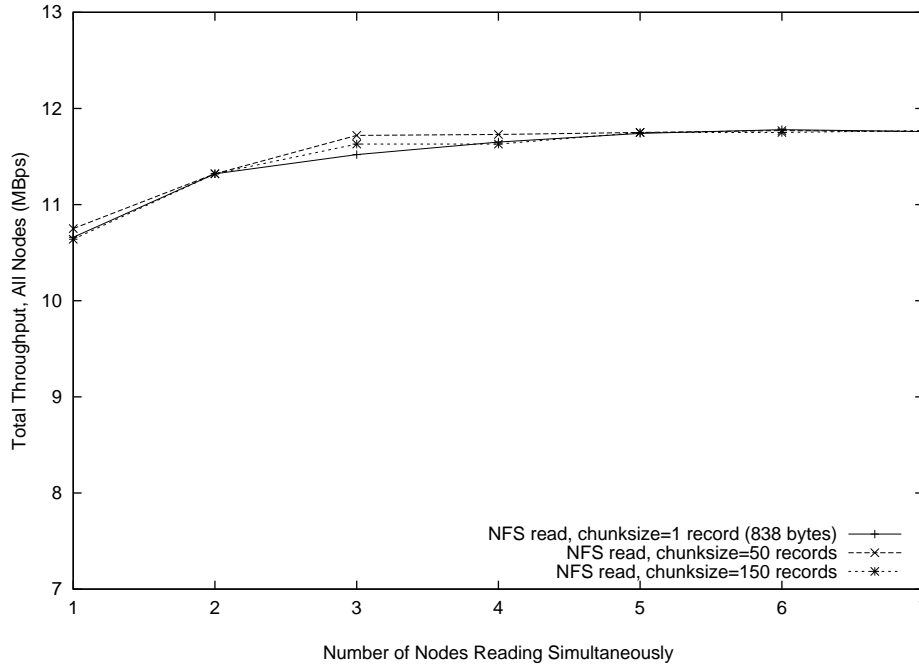


Figure 1. NFS Read, Local Whole File, Total Throughput Across All Nodes, Varying Chunksize

1. Local Whole File (LWF): Up to seven processes are spawned, one per node. Each process reads the entire file sequentially from the beginning to the end. This workload models an application in which each cluster node reads the same source file to its local memory for processing.
2. Global Whole File (GWF): Up to seven processes are spawned, one per node. Each process sequentially reads the same amount of file data, but each reads a disjoint portion of the file. From a global perspective the entire file is read, but each portion is read by a different process. This workload models an application in which each cluster node is given a partition of a file to process, and reads it into its local memory for processing.
3. Random (R): Up to seven processes are spawned, one per node. Each process calculates a random starting point and begins reading at that record in the file. This workload models an application in which one file is being processed sequentially and then indexes into a second file to retrieve a matching record. The file access in this case may have the appearance of being random, according to the index of the record that is being retrieved.

For each read experiment a configuration parameter is the amount of data read during each read operation, or the chunk size. The chunk size may be varied in the experiments from 1 record (838 bytes) up to 350 records (293,300 bytes). Also, the number of actively reading processes may be varied from one up to seven participating readers. An MPI Barrier was used at the beginning and end of each experiment to ensure consistency in the timings of the experiments.

Figures 1 through 3 illustrate the results of the Local Whole File read experiments. Figure 1 shows the result of an experiment where from one to seven processes read an entire NFS file sequentially. This models a workload in which a file is loaded into the local memory of each node. The results show that the total throughput of the system is nearly at capacity with only a single process reading from a remote NFS server. The throughput of the system is not affected by the size of the read operation. This is likely because NFS has a very sophisticated client-side cache. The client in NFS optimizes its read operations for a sequential workload, even with a very small read request size. The throughput of the system is about 10.66MBps when a single process reads one record at a time. This is somewhat less than the capacity of the Fast Ethernet network of 12.5MBps. However, the throughput only rises to 11.77MBps when seven processes are reading, which indicates that the system may be constrained by the network and communication overhead. Because of the server-side cache through the Linux buffer cache system the second

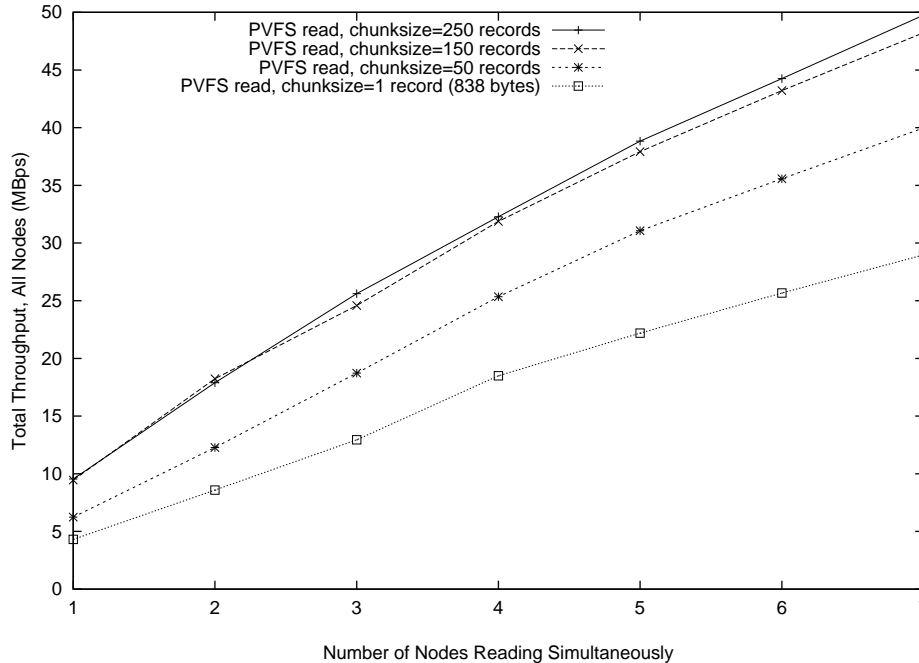


Figure 2. PVFS Read, Local Whole File, Total Throughput Across All Nodes, Varying Chunksize

read request that arrives at the server may find it in the server side cache so that the request is retrieved only from memory and transferred across the network. In that case, only the overhead from the NFS server and the network operation limit the transfer across the network.

Figure 2 illustrates the same experiment as above, but with each process reading from a PVFS file that is striped across the eight cluster nodes. Each process reads the entire file sequentially. The results show that PVFS is much more sensitive to the size of the read request. A larger stripe size results in increased throughput due to both decreased overhead and increased parallelism in the read operations. PVFS does not have any client-side caching, so each read request becomes a disk request of that size on the PVFS server and a network transfer of that size to the PVFS client. When the request is the size of a single record, the request nearly always comes from a single PVFS server. However, when the request size is greater than 10 records (8380 bytes), then the request exceeds the size of a single PVFS stripe (8192 bytes). In that case, even a single request is being served by at least two servers. As the request size grows the number of servers for a single request increases. When the request size is greater than 79 records then all eight PVFS servers participate in the file request. The results show that the performance of PVFS does not increase significantly between a request size of 150 records and a request size of 250 records.

When a single reader is reading a PVFS file, the throughput is still limited by the bandwidth of the single Fast Ethernet link into the reader node. The throughput of one node with a request size of 150 records is 9.45MBps, which is somewhat less than the throughput of the network.

Figure 3 illustrates the comparison of PVFS and NFS for the Local Whole File read, using the best available numbers for each file system. The results show that if one uses the most optimized parameters for sequential reads for this PVFS stripe size that with a single participating reader that PVFS performs nearly as well as NFS. The throughput of PVFS far exceeds that of NFS for two or more simultaneous readers. With seven participating readers the throughput of PVFS is 48.23MBps and the throughput of NFS is 11.76MBps, for a speedup of about 4.1.

Figures 4 through 6 illustrate the results of the Global Whole File read experiments. These experiments model a workload in which each cluster node reads a disjoint portion of a large file into its local memory for processing. Figure 4 shows the results of Global Whole File for NFS. As with the Local Whole File workload, NFS is relatively insensitive to the size of the read request. The combination of caching of block of data and the sequential workload causes NFS to be limited by the available network bandwidth from the NFS server. Unlike the Local Whole File workload, the performance of NFS with the Global Whole File workloads drops somewhat as the number of readers

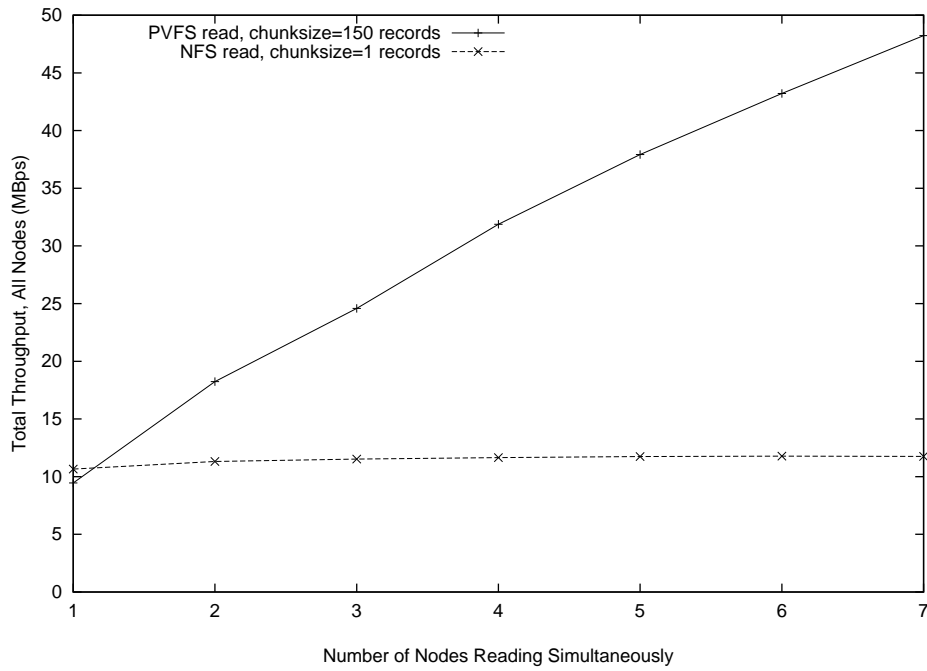


Figure 3. Best PVFS versus Best NFS Read, Local Whole File, Total Throughput

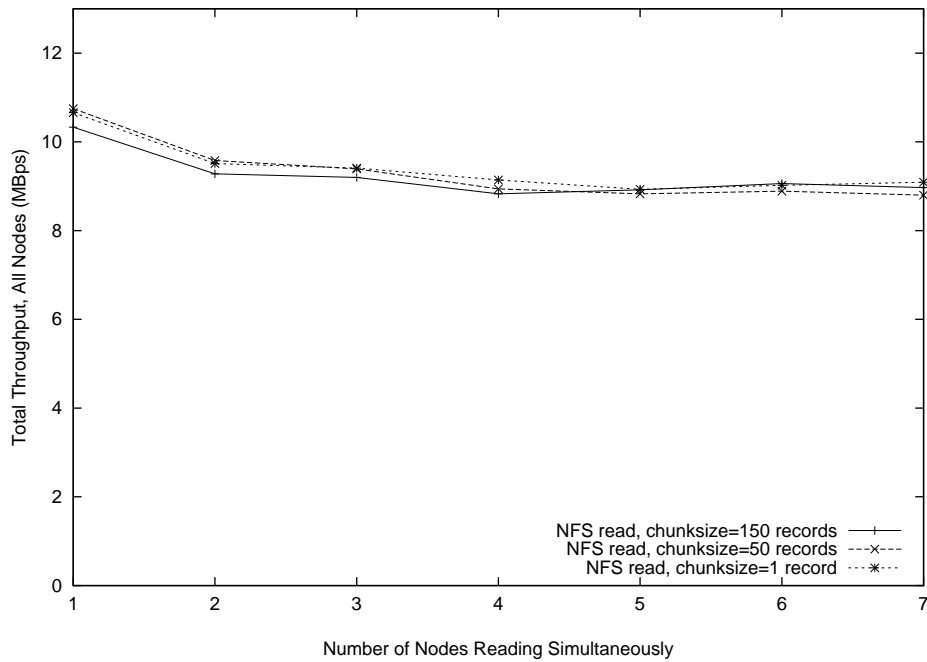


Figure 4. NFS Read, Global Whole File, Total Throughput Across All Nodes, Varying Chunksize

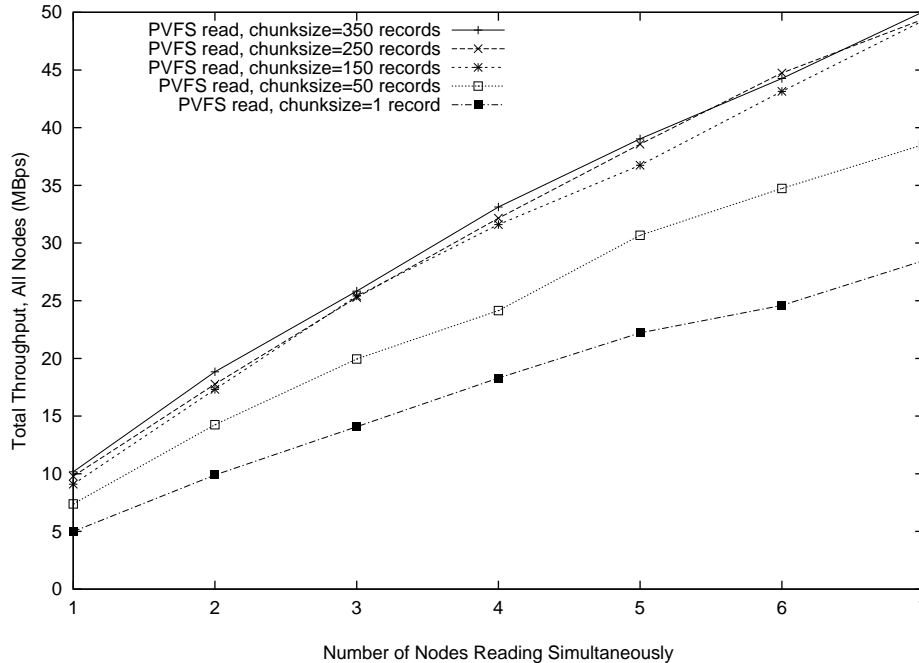


Figure 5. PVFS Read, Global Whole File, Total Throughput Across All Nodes, Varying Chunksize

increases. With each new reader, the disk subsystem on the NFS server must serve file requests that are no longer in strict sequential order. The requests to the disks are no longer contiguous, and so the disk head must scan further across the disk to retrieve file data. This extra work by the disk subsystem causes the total throughput for the workload to drop as new readers are added to the system.

Figure 5 show the results of Global Whole File for PVFS. As with the Global Whole File workload, the performance of PVFS increases as the size of the read request increases, up to the limits of the parallelism in the system. There is little difference in throughput between request sizes of 150 records, 250 records, and 350 records.

Figure 6 illustrates the comparison of PVFS and NFS for the Global Whole File read, using the best available numbers for each file system. As with Local Whole File, the results show that if one uses the most optimized parameters for sequential reads for this PVFS stripe size that with a single participating reader that PVFS performs about the same as NFS. The throughput of PVFS far exceeds that of NFS for two or more simultaneous readers. With seven participating readers the throughput of PVFS is 50.06MBps and the throughput of NFS is 9.09MBps, for a speedup of about 5.5.

Figures 7 through 9 illustrate the results of the Random read experiments. For this workload 100,000 chunks were read from the file by each participating process with varying chunk sizes. Figure 7 shows the performance comparison of NFS to PVFS for a workload that randomly retrieves a single record from a large file. Even for a single participating reader the PVFS file system performs better than NFS. This is likely because the NFS cache mechanism is working to the disadvantage of this particular workload when the chunk size is one record (838 bytes). The NFS cache mechanism retrieves a block of file data. For a workload that is sequential or has high locality this optimizes the chance that the next request is either in the cache, and optimizes the network transfer. However, for a workload that is random and where the request is smaller than the block size, a single request cause more data to be retrieved than is actually used by the application. From the application's perspective the effective throughput is lowered.

Figure 8 further illustrates the sensitivity of NFS to the size of the read request for a random workload. For this workload, the client-side cache of NFS causes the throughput to continue to rise as the request size increases. Since this workload reads a random portion of the file for 100,000 requests, as the size of the request increase, the likelihood that some portion of the request is found in the cache increases, which decreases the response time and increases the throughput of the system. It can be observed that the throughput for the Random workload for a request size of 128

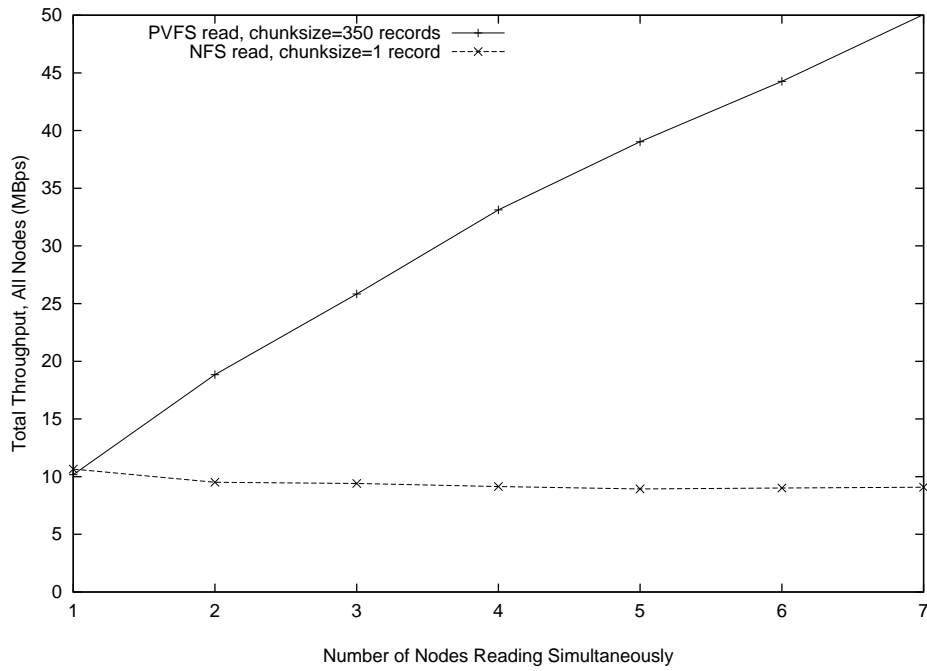


Figure 6. Best PVFS Read versus Best NFS Read, Global Whole File, Total Throughput

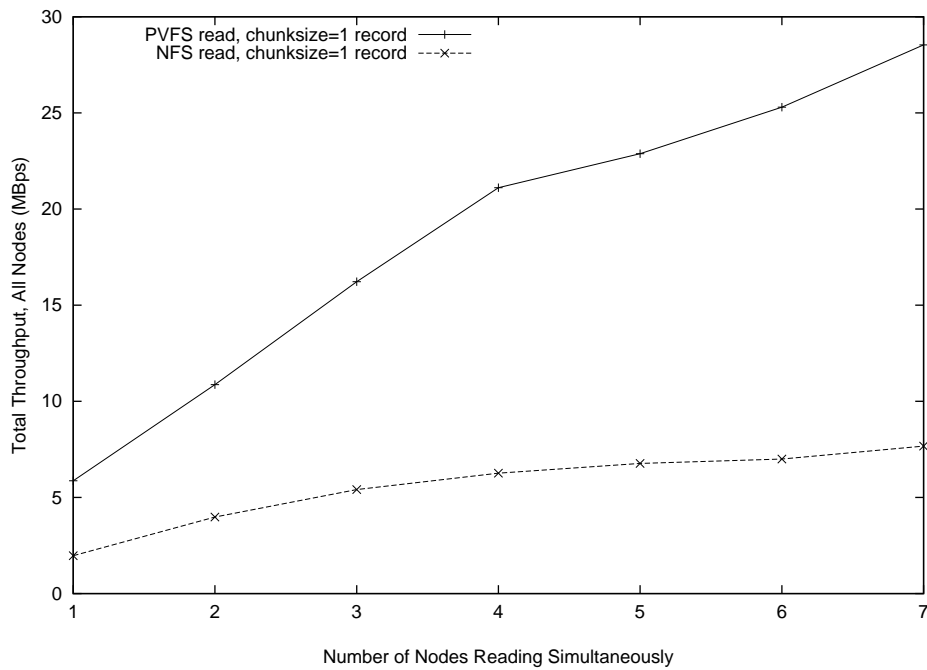


Figure 7. PVFS Random Read versus NFS Random Read, Total Throughput, Chunksize=1 record

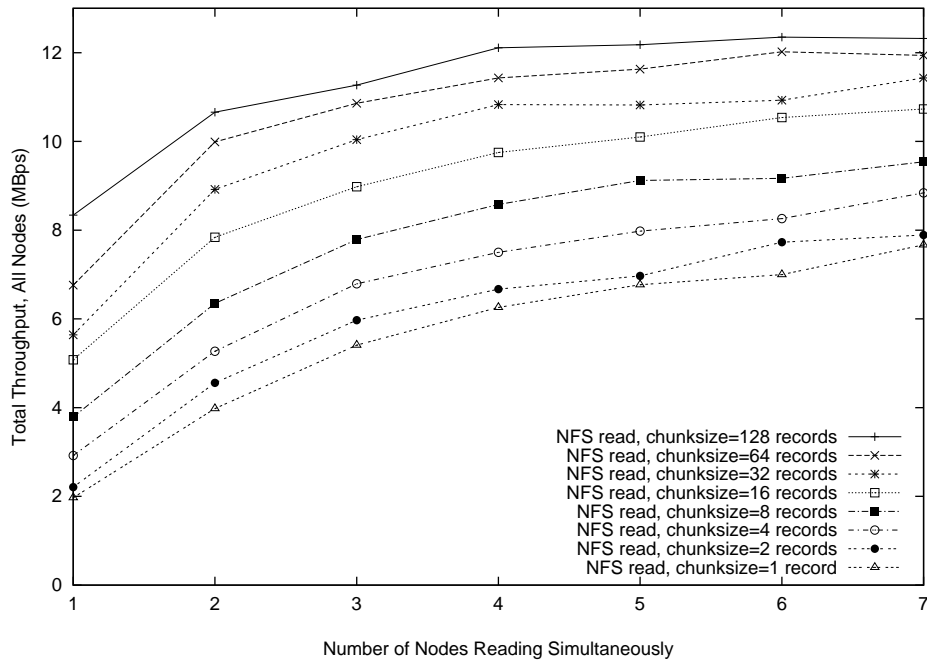


Figure 8. NFS Random Read, Total Throughput, Varying Chunksize

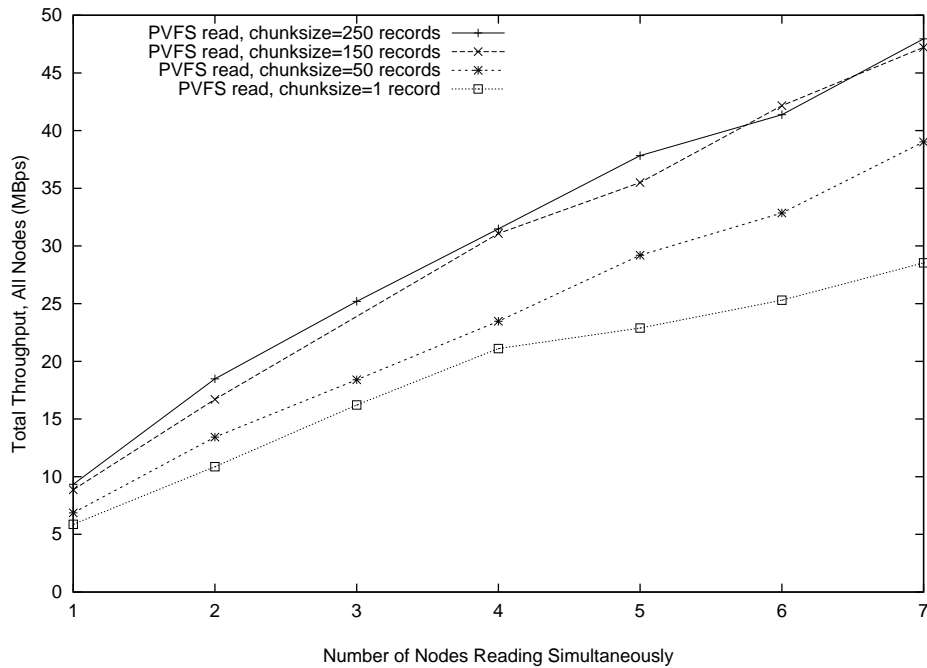


Figure 9. PVFS Random Read, Total Throughput, Varying Chunksize

records for six and seven nodes exceeds the best throughput of the Local Whole File workload. In the Local Whole File workload the request is never found in the client-size cache, whereas with the Random workload the request is sometimes found in the client-side cache. This effect will be observed particularly for experiments that run over a long period of time.

Figure 9 illustrates that PVFS is also sensitive to the size of the request for a random workload. In this case, as the size of the request approaches the sum of the stripe sizes across all eight PVFS servers, the throughput is

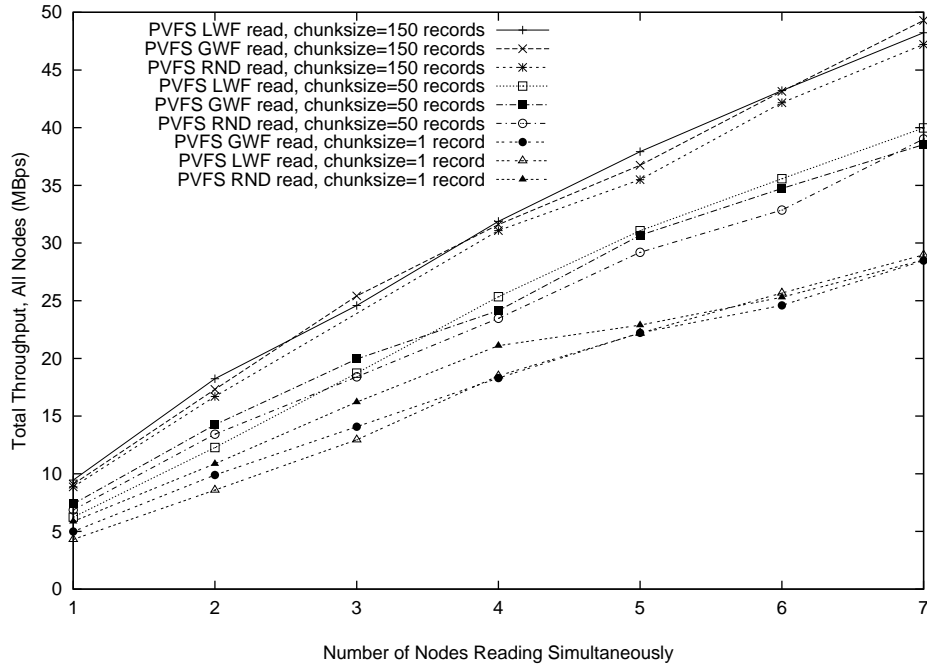


Figure 10. PVFS Read, Total Throughput, All Workloads

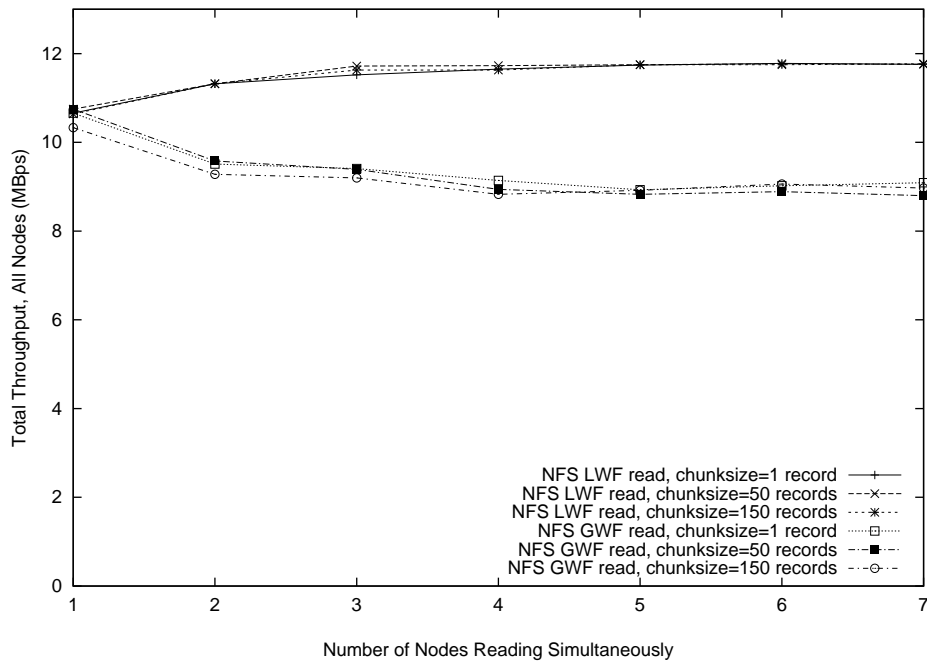


Figure 11. NFS Read, Total Throughput, All Workloads

maximized for the system. As with the Local Whole File and Global Whole File workloads, the performance of PVFS does not improve significantly when the number of records retrieved is greater than 150.

Further observations can be made from these experiments, as shown in Figures 10 and 11. Figure 10 graphs the PVFS results for all workloads, for a chunksize of 1 record, 50 records, and 150 records. As the number of participating readers increases, the throughput for each of the workloads is very similar. The results show that PVFS is more sensitive to the size of an individual read request than it is to the workload pattern, particularly for

a number of nodes greater than four.

Figure 11 graphs the NFS results for the Local Whole File and Global Whole File workloads, for 1 to 150 records. In this figure the top three lines are results for the Local Whole File workload and the bottom three lines are the results for the Global Whole File Workload. Unlike PVFS, Figure 11 shows that NFS is nearly completely insensitive to the request size for these sequential workloads, but that throughput is affected significantly by the type of read workload.

4. CONCLUSIONS AND FUTURE WORK

These performance experiments confirm that the increased parallelism for read operations that can be obtained when using a parallel file system increases the read performance in a cluster file system. However, the increase in performance is not linear. The speedup obtained by using PVFS rather than NFS in a system with seven participating readers and eight file servers is, at best, around 5.5. This suggests that performance improvements are possible, and several observations can be made.

First, the performance of NFS on our relatively modest existing cluster node hardware appears to be significantly constrained by the available switched Fast Ethernet network bandwidth. In the best case for a sequential workload in which caching is not a factor, the throughput obtained using NFS was better than 95% of the network throughput that could be delivered by the NFS server node. A faster network will likely improve the performance of NFS.

While PVFS will likely also improve in performance over a faster network, the predicted improvement is not as clear. For example, the throughput obtained for a single reader in the best case is only about 75% of the available network bandwidth. Since a single reader represents the case in which all servers are active but the throughput is limited by the network and other resources at the single reader node, this suggests that other factors are important to consider.

Second, the experiments using NFS show that performance is best when the cache is used most optimally. This is case when all of the data in the cache is used for a sequential workload, or when a request is found in the cache on a return try such as in the random workload. When only a small amount of the cache block is used, such as when a small request is made only once, then NFS performs poorly. Since PVFS does not load a cache block at a time, but rather loads the request size at a time, it out-performs NFS for a workload that has poor cache utilization. This suggests that if an application could provide hints to the file system that could describe when a workload should not cache that better performance could be obtained.

Third, PVFS performs best when each read request utilizes the maximum number of servers. This suggests that if the stripe size were customized to the application that will be reading the file that better performance could be obtained. While this may not be possible in general, for massive data processing applications of the type described in this paper it is probably possible.

PVFS lacks some items needed for a typical application that performs massive data processing. For example, the PVFS design does not provide fault tolerance. If the node containing a particular stripe is busy or unavailable then that portion of the file is delayed or cannot be accessed at all. Since massive data processing applications may execute for a long period of time, if one node is not available during a portion of the run time, then the application as it currently exists will fail. Additionally, PVFS does not have client-side caching. This makes PVFS more sensitive to a repeated read request from the client.

Our plans are to investigate client-side caching and stripe mirroring for fault tolerance in parallel file systems. In addition, we would like to develop an API that allows the application to provide hints that improve the performance of file and data access. With this API, modified cluster applications can execute even more effectively.

Through the support of Acxiom we are constructing the Acxiom Cluster Testbed (ACT) at the University of Arkansas. The ACT cluster will consist of seven dual-processor Pentium III 1GHz computers, each with 1GB memory, dual 40GB hard drives configured with RAID, and connected via Myrinet and Fast Ethernet networks. The ACT cluster will enable the testing over a high-performance cluster interconnection network.

5. ACKNOWLEDGMENTS

We would like to acknowledge the direct support of David Roland of Acxiom Corporation for funding and contribution to project ideas, and Sam Nelson and Terry Talley of Acxiom Corporation for their contribution to project ideas. We would like to acknowledge the efforts of Dan Martin and Scott Fendley of Computing Services, University of Arkansas, for their help in providing the data file that was used in the experimental testing.

REFERENCES

1. Acxiom Corporation, "GE Capital and early adoptor of Acxiom Corporation's AbiliTectm in the U.K.," in <http://www.acxiom.com>, (London), April 2001.
2. P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, "PVFS: A parallel file system for linux clusters," in *Proceedings of the 4th Annual Linux Showcase and Conference*, pp. 317–327, USENIX Association, (Atlanta, GA), 2000.
3. R. H. Arpaci-Dusseau, E. Anderson, N. Treuhaft, D. E. Culler, J. M. Hellerstein, D. Patterson, and K. Yelick, "Cluster I/O with River: Making the fast case common," in *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pp. 10–22, ACM Press, (Atlanta, GA), 1999.
4. J. Huber, C. L. Elford, D. A. Reed, A. A. Chien, and D. S. Blumenthal, "PPFS: A high performance portable parallel file system," in *Proceedings of the 9th ACM International Conference on Supercomputing*, pp. 385–394, ACM Press, (Barcelona), 1995.
5. N. Nieuwejaar and D. Kotz, "The Galley parallel file system," in *Proceedings of the 10th ACM International Conference on Supercomputing*, pp. 374–381, ACM Press, (Philadelphia, PA), 1996.
6. T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang, "Serverless network file systems," in *In Proceedings of the 15th Symposium on Operating System Principles. ACM*, pp. 109–126, (Copper Mountain Resort, Colorado), December 1995.
7. B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, "NFS version 3: Design and implementation," in *USENIX Summer*, pp. 137–152, 1994.
8. W. Gropp, E. Lusk, N. Doss, and T. Skjellum, "A high-performance portable implementation of the MPI message-passing interface standard," in *Tech. Report ANL/MCS-P5670296, Argonne National Laboratory*, February 1996.
9. C. Juszczak, "Improving the write performance of an NFS server," in *Proceedings of the USENIX Winter 1994 Technical Conference*, pp. 247–259, (San Francisco, CA, USA), 17–21 1994.
10. S. Suresh, "Performance enhancements in SunOS NFS." Technical Report TR 93-18, State University of New York, Buffalo Computer Science Dept., May, 1993.
11. D. Kotz and C. S. Ellis, "Practical prefetching techniques for multiprocessor file systems," *International Journal on Distributed and Parallel Databases* 1(1), 1993.