PVFS simulation using CODES/ROSS simulator

Sughosh Divanji*, Raghav Kapoor*, Dongfang Zhao*, Ioan Raicu*†
*Department of Computer Science, Illinois Institute of Technology, Chicago, IL,USA
†Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA
sdivanji@hawk.iit.edu, rkapoor7@hawk.iit.edu, dzhao8@iit.edu, iraicu@cs.iit.edu

ABSTRACT
HPC applications are becoming data-intensive in that they consume large amounts of data and have complex data dependencies. Current resource managers are not aware of data location (local vs remote parallel file system) and the cost to move it. In this project, as the first stage of implementing a data-aware scheduler on top of Slurm[2] resource manager with burst buffer architecture, we have simulated the PVFS[12] parallel file system for a IBM BG/P machine[22]. We have simulated the PVFS[12] parallel file system using CODES[14] and ROSS[13] parallel discrete event simulators from Argonne National Laboratory. We evaluate the performance of our simulation for I/O operations with different dataset sizes and for metadata operations. We also compare the performance of our simulation with FusionFS[11] and NFS[23] file systems. Finally, we measure the accuracy of our simulations by comparing the I/O operation throughput with actual deployment of GPFS[24] and by comparing metadata throughput with actual deployment of PVFS. The file system simulation models implemented in this project will be used to develop simulation of burst buffer architecture and eventually to simulate a data aware scheduler on top of Slurm[2] resource manager.

Categories and Subject Descriptors
D.4.8 [**Performance**]: *Simulation*

General Terms
Measurement, Performance, Design.

Keywords
HPC, data-intensive, PVFS, simulation, CODES, ROSS.

1. INTRODUCTION
Recent trend in HPC applications is that they are becoming data-intensive that consume and produce large volumes of data and have complex data dependencies. Examples of these applications are VISTA(Astronomy)[26], LIGO (Astrophysics)[27], BLAST(Bioinformatics)[28] and ATLAS(High Energy Physics)[29]. Current resource managers are ignorant of data location (local vs remote parallel file systems like PVFS[12], Lustre[30] etc) and the cost to move it. Accessing and retrieving large amounts of data is currently is just a remarkable side effect on scheduling computations.

Burst buffer architecture has been proposed to handle bursty I/O patterns in HPC systems [1]. Burst buffers are high-throughput, low-capacity storage devices that act as a staging area or a write-behind cache for HPC storage systems. The approach we follow to incorporate burst buffers is to place these buffers on I/O nodes that connect to the external storage system and to manage these buffers as part of the I/O forwarding services. If burst buffers are sufficiently large and fast, they can absorb I/O bursts. [1]. Burst buffers will integrate at HPC I/O nodes and will be managed by I/O forwarding software. By aggregating and absorbing the I/O requests into the burst buffer layer, applications can overlap computations that follow I/O bursts while asynchronously pushing data to the storage for persistence. Without the burst buffers, applications would block until all I/O requests are completed and would allow no potential for optimization or overlapping computation and I/O activity.

Before we simulate burst buffer architecture and measure its effect on scheduler performance we need to simulate existing parallel file system on a HPC machine. So, in this project we simulate PVFS[12] file system on a IBM BG/P architecture[22]. PVFS is a open-source parallel file system developed by Argonne National Lab and Clemson University. In the PVFS architecture the compute nodes act as clients and the I/O nodes act as servers. The application resides on the client and triggers I/O requests to the PVFS client daemon running on the client. The PVFS client daemon communicates with the PVFS server daemon running on the servers to handle I/O operations and metadata operations. The servers are again classified into I/O servers which handle I/O operations and metadata servers which handle metadata operations. Originally PVFS had a single centralized metadata server and multiple distributed I/O servers. PVFS2 has a distributed metadata architecture. Each server can either function as a I/O server or a metadata serevr or both. The server type is configured in software. PVFS also implements a file striping mechanism, with each file divided into multiple stripes with each stripe being stored in a different disk. The file distribution information includes both the file location and location of the disk in the cluster. The location of the file is specified with three parameters, base I/O node number, number of nodes and the stripe size. We have implemented our simulation using CODES(Co-Design of Multilayer Exascale Storage Architectures)[14] and ROSS(Rensselaer Optimistic Simulation System)[13] simulation frameworks from Argonne National Laboratory and Rensselaer Polytechnic Institute. ROSS is a parallel discrete event simulator which uses time warp protocols[31] to simulate discrete events in parallel. ROSS also allows reverse computations to rollback changes for timestamp mismatches. Because of this feature ROSS can achieve very high performance as it can execute events in parallel. CODES is built on top of ROSS and provides various network models for torus and dragonfly networks and supports MPI collective communication operations

using the optimistic event scheduling capability of ROSS. We measure the performance of our simulation for I/O operations and metadata operations. We have compared our simulation with FusionFS[11] and NFS[23] file systems through simulations. We have also measured the accuracy of our simulations by comparing the read and write throughput of our simulation with actual GPFS read and write throughput and by comparing the metadata throughput of our simulation with metadata throughput of actual PVFS[12] implementation[11].

The rest of the paper is organized as follows. In section 2, we explain our proposed solution. In section 3, we present the evaluation of our solution. In section 4, we present the related work. In section 5 we present our future work and conclude the paper.

2. PROPOSED SOLUTION

As a first stage of implementing a data aware scheduler using a burst buffer architecture [1] we simulate PVFS file system[12] on a IBM BG/P[22] machine.

The architecture of our simulation is as below shown in Figure 1.

In CODES/ROSS the basic unit of simulation is a Logical Processor (LP). LPs are abstractions of simulated physical processes. They act like real processes in the system and are synchronized by Time Warp protocol[31]. In our simulations we simulate each node as a LP and the arrows which indicates communication between different nodes are simulated as events taking place between the LPs.

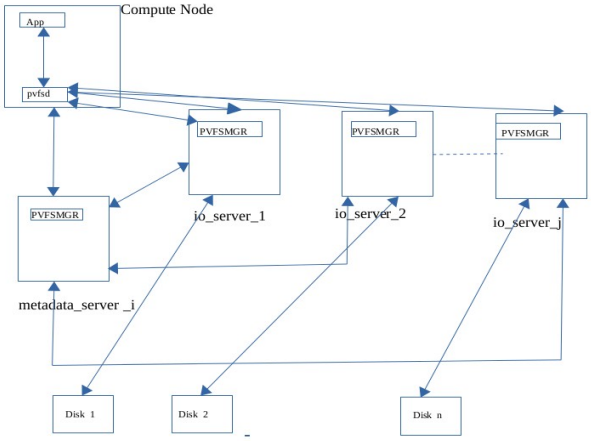

Figure 1: PVFS simulation architecture.

Application resides on the compute node and triggers the I/O request to the PVFS client daemon running on the compute node. We model the client node in client LP in our simulation. The servers are modeled in the server LP in our simulation. PVFS servers are of two types: metadata servers which handle metadata of the files and I/O servers which handle actual application data. In PVFS the metadata server and I/O server config is done in software. A single I/O node can act as either a metadata server or I/O server or both. In our simulation we assume that every I/O node is acting as both metadata server and I/O server. We consider disk as an overhead for the operations done in our simulator and we will take up modeling disk as a separate LP in our future work.

In the sections 2.1 to 2.4 we describe the various operations in PVFS[12].

2.1 File Create

The file create operation takes as shown in Figure 2 below.

The steps for file creation are:

1. Application on compute node sends file_create() request to pvfsd running on compute node.
2. pvfsd running on compute node sends create_metafile() request to pvfsmgr running on a random metadata server, say metadata_server_i.
3. pvfsmgr running on metadata_server_i returns metadata handle through return_metadata().
4. pvfsd on compute node sends create_datafile() requests to pvfsmgr on a set of IO servers.
5. Each io server receiving create_datafile() sends create_dir_entry() to the metadata_server_i.
6. metadata_server_i sends metadata attributes to the io servers through set_meta_attr()
7. Each io server returns data handle to pvfsd on compute node through return_datahandle().
8. pvfsd informs application that file is created using file_create_done().

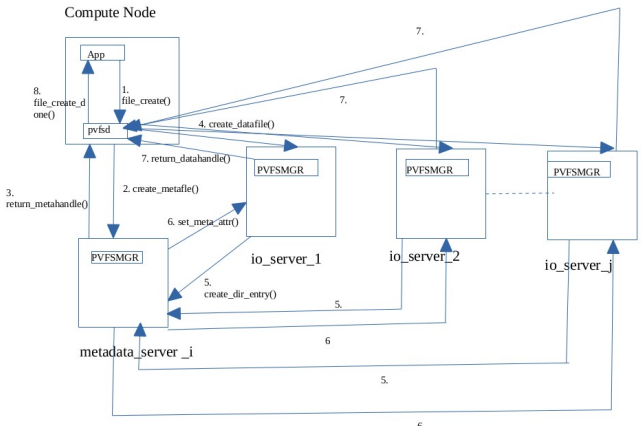

Figure 2 : File Create.

2.2 File Open

The file open operation takes as shown below in Figure 3.

The steps for file open are as follows:

1. Application on compute node sends a file_open() request to pvfsd on compute node.

2. pvfsd on compute node sends a lookup_request() to all metadata servers.
3. The metadata server which has metadata for the file, say metadata_server_i returns file handle through return_file_handle().
4. pvfsd on compute node sends request_datafile() to all io servers which have the data handle for the file.
5. Each io server returns the handle for the stripe it is handling to pvfsd on compute node.
6. pvfsd informs the application that the file has been opened through file_open_done().

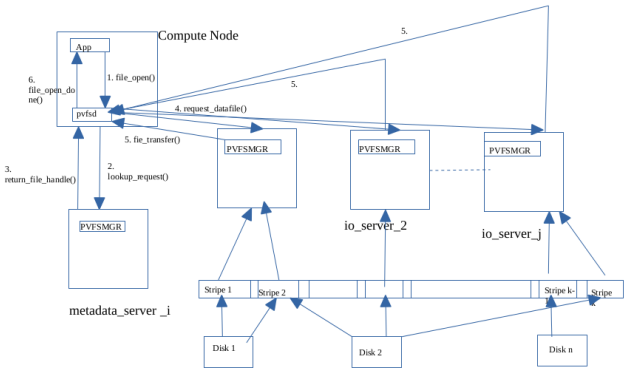

Figure 3: PVFS file open.

2.3 PVFS file read

In PVFS, file read operation takes place as shown below in Figure 4.

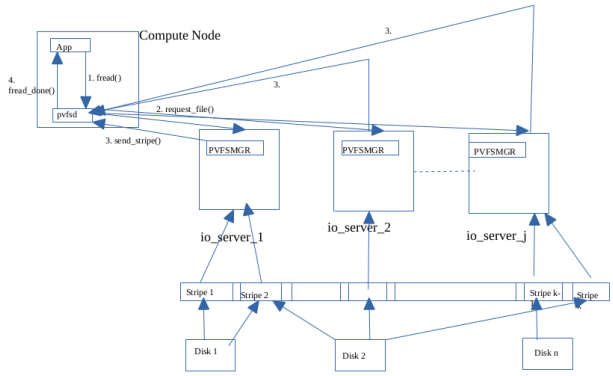

Figure 4: PVFS file read.

The steps for file read as follows:

1. Application on compute node sends fread() request to pvfsd on compute node.

2. pvfsd sends request_file() to each io server that has the data handle for the file.

3. Each io server sends the stripe it is handling to the pvfsd on compute node.

4. pvfsd informs the application that file read is done through fread_done().

2.4 PVFS File write

PVFS file write operation is done as shown in Figure 5.

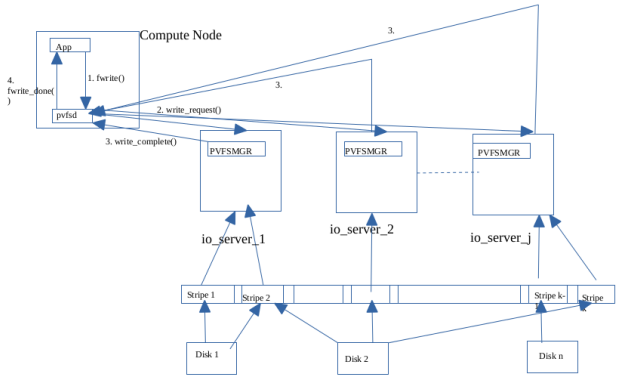

Figure 5: PVFS file write

The steps for file write as follows:

1. Application on compute node sends fwrite() request to pvfsd on compute node.
2. pvfsd sends write_request() to io server.
3. pvfsmgr on io server informs pvfsd that write is successfully completed through write_done()
4. pvfsd informs application that write is succesfully completed through fwrite_done().

We have developed our simulation model in multiple stages. In the first stage, We have developed and evaluated a hardcoded model for PVFS file system in which each node acts as both server and client . From the CODES/ROSS simulation perspective our simulation model consists of only one Logical Processor (LP). We assume all remote I/O operations as per the architecture of PVFS. We assume a constant I/O overhead of 5% for each operation and add this overhead to the time taken to complete each operation. We also take this value as the seed for random I/O noise and add the I/O noise for the execution time. We assume a constant metadata size of 32 bytes. We have measured the time taken to complete I/O operations with different dataset sizes and different number of nodes. From this value we calculate the aggregate throughput.

In the second stage we have separated the client and server nodes. As in the first stage, we assume all remote I/O operations as per the architecture of PVFS. We assume a constant I/O overhead of 5% for each operation and add this overhead to the time taken to complete each operation. We also take this value as the seed for random I/O noise and add the I/O noise for the execution time. We assume a constant metadata size of 32 bytes. In our simulation, the

server nodes communicate with the client nodes in a round-robin fashion. In the configuration file used by the simulator we specify the data size to be transferred in each iteration through a variable called pvfs_file_size. By default this value is set to 64KB which is the default stripe size in PVFS[32]. We repeat this workload for a large number of iterations. So, effectively we are sending one stripe in each round of communication. In this way we have simulated PVFS file striping. We have simulated an event called request event in which we send a single stripe in each iteration from client to server and repeat this workload for a large number of iterations. Hence, this event simulates file write. We have simulated another event called ack event in which we send a single stripe in each iteration from server to client and repeat this workload for a large number of iterations. Hence, this event simulates file read. Since, both request and ack events happen in the same round of communication, file read and write operations are simulated in a single round of communication. We have another variable called payload_size in the configuration file which is set to 32 bytes and this represents the file metadata. We simulated metadata operations by sending payload_size as the data size in the simulator event and repeating this workload for only one iteration.

We have measured the time taken to complete I/O operations and metadata with different dataset sizes and different number of server nodes with the number of client nodes fixed at 1024. From this we calculate the aggregate throughput for I/O operations and metadata operations. We have also compared the performance of PVFS[12] with FusionFS[11] and NFS[23] file systems. We have evaluated the performance of our simulation in terms of accuracy by comparing the I/O throughput to actual GPFS[24] read and write throughput from FusionFS paper[11] and comparing the metadata performance to actual PFVS metadata performance from FusionFS paper[11].

2.5 Implementation Details

We have implemented our code in C. The total project code is around 550 lines of C code. The CODES/ROSS framework consists of around 45K lines of C/C++ code. We have used version 0.40 for codes-base and codes-net libraries of CODES simulator and for the commit hash of ROSS is 44b7b9a which is the latest version of ROSS at the time of release of CODES 0.40. The code for the project is available at https://github.com/sdivanji/pvfs_sim/

3. EVALUATION

3.1 Testbed

We have run our simulations on a single machine with a AMD 4-core, 64-bit processor and 8GB of DDR3L RAM. We have used MPICH2[25] version 3.04 from ANL as our MPI library. We have run all experiments in the sequential mode of ROSS as we have not implemented reverse computations in our code. Since, our simulations do not take a lot of time to run, the effort needed to code the reverse computation is very high when compared to its benefits.

3.2 Metrics

We have evaluated the performance of our simulations in terms of Throughput(MB/s) and execution time(s) for read and write operations and Throughput(Number of operations per second) for metadata operations. We also compare the performance of PVFS[12] with respect to FusionFS[11] and NFS[23] in terms of Throughout(MB/s) and execution time(s) for read and write operations. We compare the performance of PVFS[12] with FusionFS[11] for metadata operations in terms of Throughput(Number of operations per second). We have measured the accuracy of our simulation by comparing read and write throughputs with actual GPFS read and write throughput of GPFS[24] file system in terms of MB/s and by comparing metadata throughput with PVFS[12] metadata throughput with data obtained from FusionFS BigData 2014 paper[11].

3.3 Experiments

3.3.1 Throughput

We first evaluate the performance of our first stage model in terms of throughput and execution time for read and write operations. We vary the number of nodes from 8 to 64 and the dataset size from 64MB to 1GB which is equally divided among all nodes. We use a 3D torus network model for these experiments.

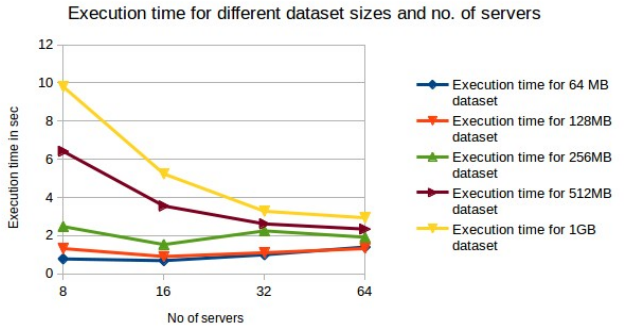

Figure 6: Execution time for different datset sizes and different number of nodes.

From the graphs we can see that at lower file sizes, execution time increases and throughput decreases with increase in the no. of nodes because the overhead of network communications is large when compared to file size. However, for larger file sizes the cost of network communication gets amortized and we see improved performance with increase in the number of nodes.

Next we evaluate the performance of our second stage model. We measure the performance in terms of execution time and throughput for read and write operations. We fix the number of clients at 1024 and vary the number of servers from 8 to 64. For each client we vary the file size from 64MB to1GB. So, in total our system will have dataset size varying from 64GB to 1TB. We have used simplenet network model available in CODES as the torus

network model does not scale for more than 8 servers and 32 clients. The results are as shown in Figure 8 and Figure 9.

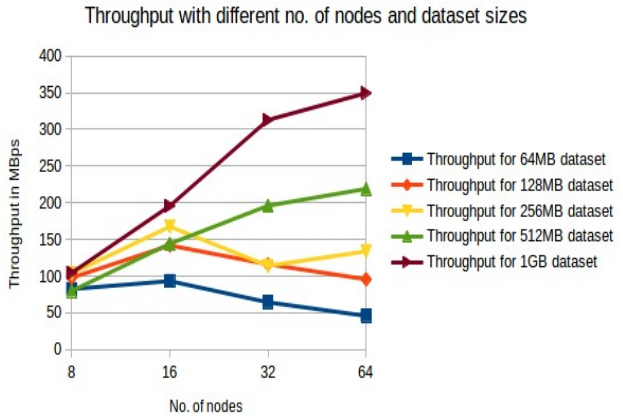

Figure 7: Throughput for different dataset sizes and different number of nodes.

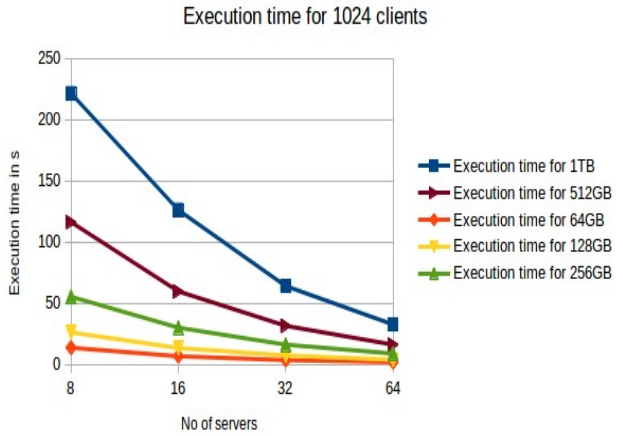

Figure 8: Execution time for 1024 clients with different number of servers and different dataset sizes.

From the graphs we can see that at larger dataset sizes the cost of remote communication is small enough that it gets amortized and we get a linear scalability with the increase in the number of servers.

From the graphs we can see that at larger dataset sizes the cost of remote communication is small enough that it gets amortized and we get a linear scalability with the increase in the number of servers.

From these experiments we can conclude that PVFS is more suited to handle larger dataset sizes as it scales almost linearly at larger dataset sizes.From these experiments we can conclude that PVFS is more suited to handle larger dataset sizes as it scales almost linearly at larger dataset sizes.

3.3.2 Ideal Stripe Size

Next we evaluate the performance in terms of throughput by varying the stripe size. We keep the number of servers fixed at 64, number of clients fixed at 1024 and the dataset size in the file system at 64GB. We vary the stripe size from 1KB to 1MB and measure the throughput for each stripe size. The results are as shown below in Figure 10.

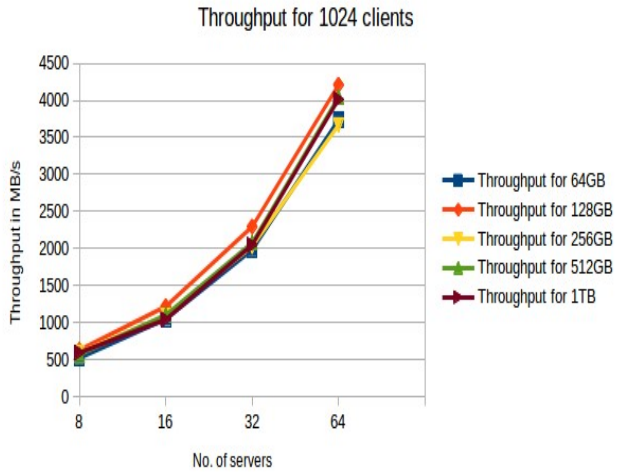

Figure 9: Throughput for 1024 clients with different number of servers and different dataset sizes.

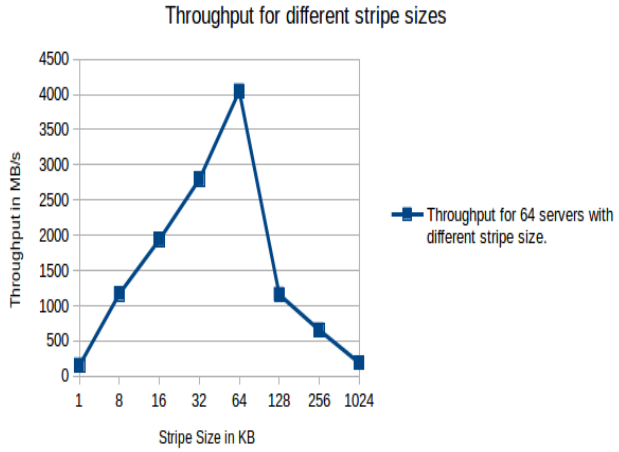

Figure 10: Throughput for different stripe sizes.

From the graph we can see that at very low stripe sizes, the throughput is low. This is because to perform read and write operations we need many cycles of communication between clients and servers. The overhead of this reduces the throughput. The throughput improves with increase in stripe sizes till 64KB as the clients need to make less number of requests to the servers. However after 64KB the throughput keeps on dropping as we increase the stripe size. This is because the packet size of the client and server networks is very less compared to the stripe size. The client network has a packet size of 8KB and the server network has a packet size of 2KB. So, at higher stripe sizes

each stripe consists of larger number of packets and this packetization overhead reduces the throughput at higher stripe sizes.

From this experiment we conclude that 64KB is the ideal stripe size. The real PVFS 2.0 deployments also have 64KB as the ideal stripe size[32]. Our simulations confirm this.

3.3.3 Metadata

We measure the metadata performance of our PVFS simulation in terms of operations per second. To simulate the metadata operations we send very small amount of traffic (32 bytes) through our models and measure the number of operations per second. We keep the number of clients fixed at 1024 and vary the number of servers from 1 to 64. The results are as shown below in Figure 11.

From the figure we can see that the metadata performance saturates at 32 servers. This is because the data being transferred is so small that the overhead of adding additional nodes saturates the performance.

3.3.4 Throughput Comparison

We compare the performance of different file systems namely FusionFS[11] , PVFS and NFS[23]. These file systems represent different types of file systems with FusionFS for distributed file systems, PVFS for parallel file systems and NFS for centralized file systems. So, this experiment also gives us insight into the performance of different file system architectures.

We compare the performance of the file systems in terms of throughput for I/O operations for different file systems. For FusionFS we measure the throughput at 1024 nodes, for PVFS we meausre the throughput for 64 servers and 1024 clients and for NFS we measure the throughput with 1 server and 1024 clients. We vary the dataset size from 64GB to 1TB. The results are as shown below in Figure 12.

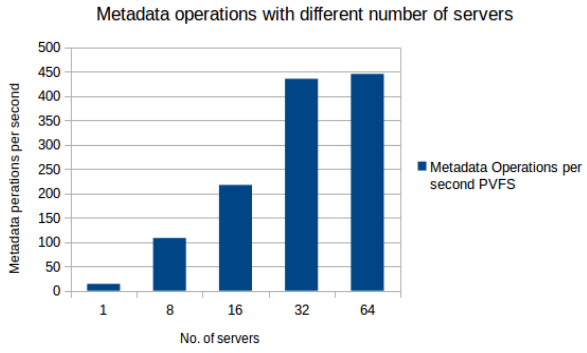

Figure 11: Metadata performance with different number of servers.

The Y-axis is plotted in a logarithmic scale to better show the difference in performance.

From the graph we can see that FusionFS outperforms PVFS and NFS at all scales. This is because FusionFS has a fully distributed architecture with each node capable of running the client and server proesses. Thus, there is a 1:1 mapping between clients and servers. Also, FusionFS enables write locality. So, FusionFS performs much better than PVFS and NFS.

PVFS follows a parallel architecture with number of servers an order of magnitude less than the number of clients. Also, all writes are remote in PVFS. NFS, on the other hand has a centralized architecture with a single centralized server. Hence, NFS doesn't scale very well and though PVFS outperforms NFS its performance is much less when compared to FusionFS.

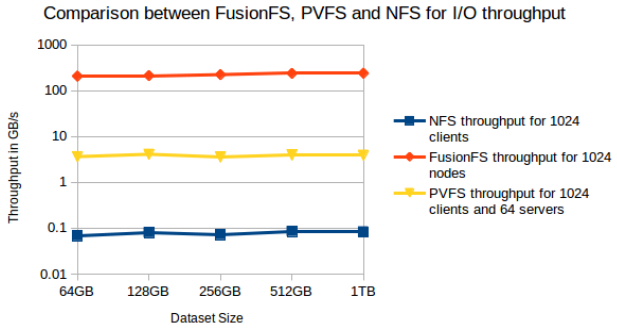

Figure 12: Comparison between PVFS, NFS and FusionFS for I/O throughput.

3.3.5 Metadata Comparison

We also compare the metadata performance of PVFS with FusionFS in terms of number of operations per second. We simulate PVFS metadata performance by sending very less amount of traffic(32 bytes) through our models. We keep a 1-1 mapping between clients and servers and vary the number of nodes from 1 to 64 for both PVFS and FusionFS so that we keep similarity in the workloads to compare. For FusionFS we use the data from FusionFS paper[11].

The results are as shown below in Figure 13.

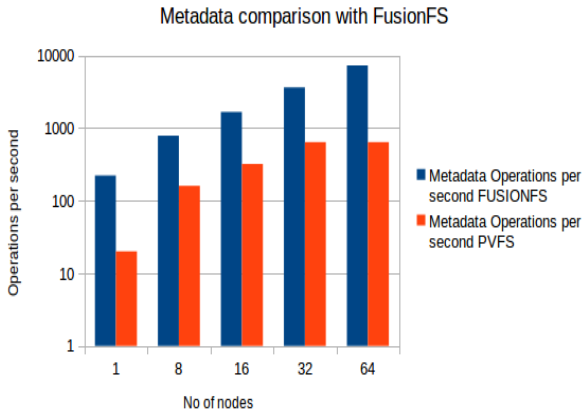

Figure 13: Metadata performance comparison between FusionFs and PVFS.

The Y-axis is in logarithmic scale to better show the difference in performance.

From the graph we can see that FusionFS outperforms PVFS at all scales. This is because FusionFS implements metadata optimization operations like update->append[11] which are highly effective. On the other hand, in PVFS no metadata optimization operations are implemented. Thus, this experiment shows that only increasing the number of metadata servers does not necessarily improve the ability to handle higher concurrency.

3.3.6 Throughput Accuracy

We measure the accuracy of our simulations by comparing throughput for read and write operations and metadata performance with actual parallel file system implementations. We compare read and write throughput of our simulation with GPFS[24] deployed on IBM Blue Gene[22]. We fix the number of servers at 128 and vary the number of clients from 1 to 1024. The results are as shown below in Figure 14 and 15.

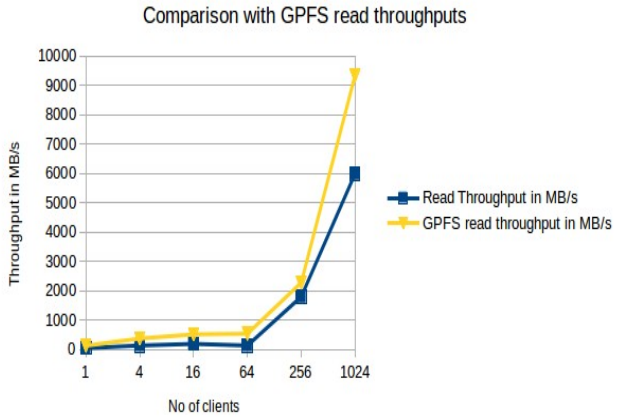

Figure 14: Comparison of read throughput with GPFS .

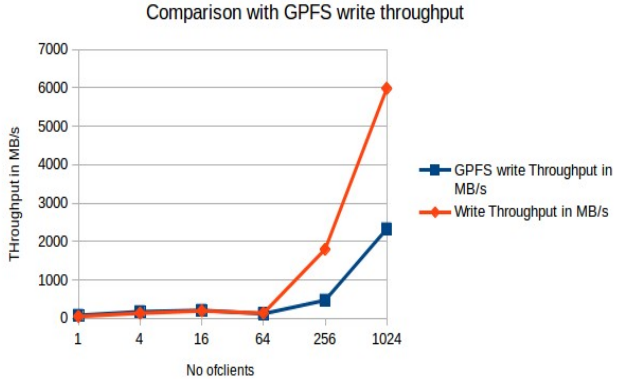

Figure 15: Comparison of write throughput with GPFS

From these experiments we can see that at lower scales our simulation matches closely with actual deployments. At higher scales there is a difference between our results and actual deployments. We attribute this difference to the fact that we were not able to use torus network models at higher scales as we ran out of memory and had to use simplified network models at larger scales.

3.3.7 Metadata Accuracy

We have compared the accuracy of our simulation for metadata performance by comparing the number of operations per second obtained from our simulation with actual PVFS deployment. We keep a 1-1 mapping between clients and servers and vary the number of nodes from 1 to 64. The results are as shown in Figure 16.

From the graph we see that at higher scales the performance of our simulations resemble closely with that of the actual deployment. At lower scales there is a difference in performance because, in our simulations we add random I/O noise and this creates a few stragglers. The effect of the stragglers is more prominent at lower scales and brings down the performance of our simulation.

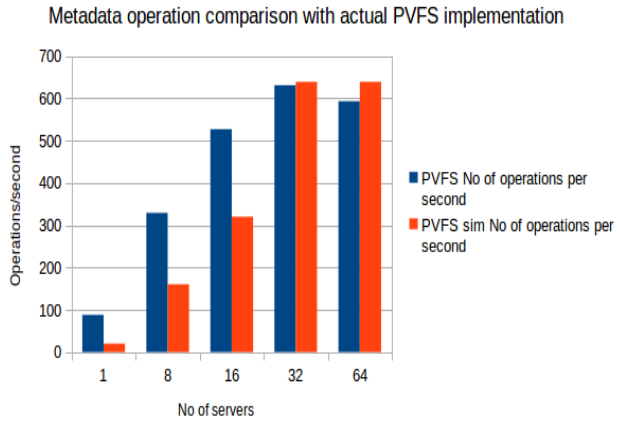

Figure 16: Comparison of metadata performance with actual PVFS implementation.

3.3.8 Effect of adding more clients

We have measured the effect of adding more clients keeping the number of servers constant. We keep the number of servers fixed at 8 and vary the number of clients from 1 to 32 and measure the I/O throughput. The results are as shown below in Figure 17.

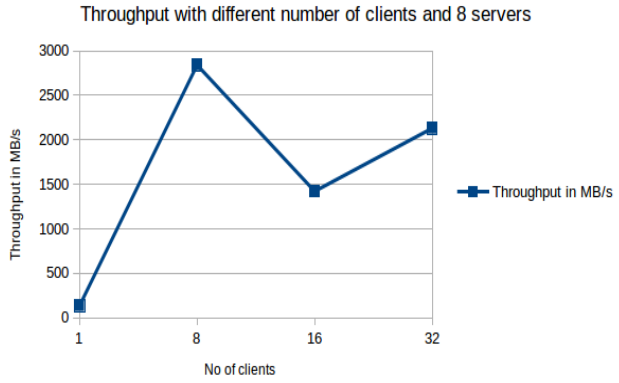

Figure 17: Different number of clients with servers fixed at 8.

From the graph we can see that when the number of clients is less than the number of servers, some servers are

underutilized and throughput is low. We can also see that when the clients and servers are in a 1:1 mapping we get the best performance. At scale of 16 clients 50% of nodes in become stragglers and this brings down the performance when compared to 24 clients scale which has less than 25% of nodes as stragglers.

4. Related Work

As part of studies going on exascale design, there is significant interest in understanding how parallel system software such as MPI/MPI-IO and the associated supercomputing applications will scale on future architectures. For example, Perumalla's μπ system [33] will allow MPI programs to be transparently executed on top of the MPI modeling layer and simulate the MPI messages. μπ has executed MPI jobs with over 27 million tasks which was executed on 216,000 Cray XT5 cores.[33] A number of universities and national labs have together developed the Structural Simulation Toolkit (SST) [34]. SST includes a collection of hardware component models including processors, memory and network at different accuracy. These models use parallel component-based discrete event simulation based on MPI. The users are able to leverage multi-scale nature of SST by trading off between accuracy, complexity, and time to solution. BigSim [35] focuses on the model and prediction of sequential execution blocks of large scale parallel applications. The model is trace-driven and it uses the scalable trace gained from machine learning for predicting overall performance. While our simulator accurately captures the large-scale parallel file system characteristics, these systems are more focused on providing accurate, large-scale computational performance models. Researchers have also developed a number of parallel file system simulators. The IMPIOUS simulator [36] was developed for fast evaluation of parallel file system designs. It simulates PVFS, PanFS, and Ceph file systems based on user-provided file system specifications, including data placement strategies, replication strategies, locking disciplines, and caching strategies. The HECIOS simulator [37] is an OMNeT++ simulator for PVFS. HECIOS was used to evaluate scalable metadata operations and file data caching strategies for PVFS. PFSsim [38] is an OMNeT++ PVFS simulator that allows researchers to explore I/O scheduling algorithm design. PVFS and ext3 file systems have been simulated using colored Petri nets [39]. This simulation method yielded low simulation error, with less than 10% error reported for some simulations. The focus of CODES sets it apart from these related simulation tools. One of the goals of CODES is to accurately and quickly simulate large-scale storage systems. To date, CODES has been used to simulate up to 131,072 application processes, 9512 PVFS file system clients, and 123 PVFS file servers [19]. The existing simulators limited their simulations to smaller parallel systems (up to 10,000 application processes and up to 100 file servers).

Research has also been done in simulating PVFS using older versions of CODES and ROSS. Ning Liu et al [19] and Bo Feng at al [20] have developed simulation models to simulate PVFS[12] file system on a IBM Blue Gene[22] machine. The difference in our work is that these simulations use their own simplified network models. We build our simulations on top of latest version of CODES[14] which is built on top of ROSS and provides realistic network models like Torus, Dragonfly etc.

5. Conclusion and Future Work

In this paper we present a simulation of PVFS[12] parallel file system using CODES[14]/ROSS[13] simulator from ANL. We measure the performance of our simulation in terms of throughput in MB/s for I/O operations and number of operations per second for metadata operations. We compare the performance of PVFS[12] with FusionFS[11] and NFS[23] file systems for I/O operations throughput and with FusionFS[11] file system for metadata operations through simulations. We measure the accuracy of our simulation by comparing it with real deployment of GPFS[24] file system for read and write operations and by comparing with real deployment of PVFS[12] file system for metadata operations. This work will act as a first stage in the implementation of a data aware scheduling system which will be built on top of slurm[2] resource manager and will use burst buffer architecture[1].

In future we plan to run the simulations with torus network models on large scale on a cluster to get more accurate results. We plan to separate disk into a separate LP so that network latency of remote disks is also considered. The next step in this project is to simulate burst buffer models and measure the effect of adding burst buffers on performance. Once, this is done we will simulate data aware scheduling on top of slurm[2] resource manager.

6. References

[1] Liu, Ning, et al. "On the role of burst buffers in leadership-class storage systems." *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*. IEEE, 2012.

[2] Yoo, Andy B., Morris A. Jette, and Mark Grondona. "Slurm: Simple linux utility for resource management." *Job Scheduling Strategies for Parallel Processing*. Springer Berlin Heidelberg, 2003

[3] B. Van Essen, R. Pearce, S. Ames, and M. Gokhale. On the role of NVRAM in data-intensive architectures: an evaluation. In International Symposium on Parallel and Distributed Processing (to appear), 2012.

[4] N. Master, M. Andrews, J. Hick, S. Canon, and N. Wright. Performance analysis of commodity and enterprise class flash devices. *In Proceedings of the 5th Parallel Data Storage Workshop (PDSW'10), November 2010.*

[5] S. Alam, H. El-Harake, K. Howard, N. Stringfellow, and F. Verzelloni. Parallel I/O and the metadata wall. *In Proceedings of the 6th Parallel Data Storage Workshop (PDSW'11), November 2011.*

[6] J. He, J. Bennett, and A. Snavely. DASH-IO: and empirical study of flash-based IO for HPC. *In Proceedings of TeraGrid'10, August 2010.*

[7] L. Gomez, M. Maruyama, F. Cappello, and S. Matsuoka. Distributed diskless checkpoint for large scale systems. *In Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing (CCGrid'10), May 2010.*

[8] P. Nowoczynski, N. Stone, J. Yanovich, and J. Sommerfield. Zest: Checkpoint storage system for large supercomputers. *In 3rd Petascale Data Storage Workshop, November 2008.*

[9] J. Bent and G. Grider. Usability at Los Alamos National Lab. *In The 5th DOE Workshop on HPC Best Practices: File Systems and Archives, September 2011.*

[10] K.Wang, X.Zhou, K.Qiao, M.Lang, B. McClelland, I. Raicu. Slurm++: a Distributed Workload Manager for Extreme- Scale High-Performance Computing Systems.

[11] D Zhao, Z Zhang, X Zhao, T Li, K Wang, D Kimpe, P Carns, R Ross, I Raicu. FusionFS: Toward Supporting Data-Intensive Scientific Applications on Extreme-Scale High-Performance Computing Systems. *In IEEE BigData 2014* .

[12]. P Carns, W Lingon, R Ross, R Thakur, PVFS: A Parallel File System for Linux Clusters. *In Proc. of the Extreme Linux Track: 4th Annual Linux Showcase and Conference, October 2000.*

[13] Carothers, Christopher D., David Bauer, and Shawn Pearce. "ROSS: A high-performance, low-memory, modular Time Warp system." *Journal of Parallel and Distributed Computing* 62.11 (2002): 1648-1669.

[14] Mubarak, Misbah, et al. "Using massively parallel simulation for MPI collective communication modeling in extreme-scale networks." *Proceedings of the 2014 Winter Simulation Conference*. IEEE Press, 2014.

[15] K. J. Bowers, B. J. Albright, L. Yin, B. Bergen, and T. J. T. Kwan. Ultrahigh performance three-dimensional electromagnetic relativistic kinetic plasma simulation. Physics of Plasmas, 15(5):7, 2008.

[16]C Nieter,JR Cary. VORPAL: a versatile plasma simulation code - Journal of Computational Physics, 2004 – Elsevier

[17] https://www.alcf.anl.gov/projects/climate-weather-modeling-studies-using-prototype-global-cloud-system-resolving-model-0

[18]Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 24/7 characterization of petascale I/O workloads. In *Proceedings of 2009 Workshop on Interfaces and Architectures for Scientific Data Storage*, September 2009.

[19]Liu, N., Carothers, C., Cope, J., Carns, P., Ross, R., Crume, A., & Maltzahn, C. (2011, September). Modeling a leadership-scale storage system. In*Proceedings of the 9th international conference on Parallel Processing and Applied Mathematics-Volume Part I* (pp. 10-19). Springer-Verlag.

[20]Feng, Bo, Ning Liu, Shuibing He, and Xian-He Sun. "HPIS3: towards a high-performance simulator for hybrid parallel I/O and storage systems." In*Proceedings of the 9th Parallel Data Storage Workshop*, pp. 37-42. IEEE Press, 2014.

[21] Haddad, Ibrahim F. "Pvfs: A parallel virtual file system for linux clusters." *Linux Journal* 2000.80es (2000): 5.

[22]Gara, Alan, et al. "Overview of the Blue Gene/L system architecture." *IBM Journal of Research and Development* 49.2.3 (2005): 195-212

[23]Sandberg, Russel. "The Sun network file system: Design, implementation and experience." *Distributed Computing Systems: Concepts and Structures* (1987): 300-316

[24]Schmuck, Frank B., and Roger L. Haskin. "GPFS: A Shared-Disk File System for Large Computing Clusters." *FAST*. Vol. 2. 2002

[25]Gropp, William. "MPICH2: A new start for MPI implementations." *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer Berlin Heidelberg, 2002. 7-7.

[26]Emerson, J., and W. Sutherland. "The Visible and Infrared Survey Telescope for Astronomy (VISTA): Looking Back at Commissioning." *The Messenger* 139 (2010): 2-5.

[27]Abbott, B. P., et al. "LIGO: the laser interferometer gravitational-wave observatory." *Reports on Progress in Physics* 72.7 (2009): 076901

[28]Oehmen, Chris, and Jarek Nieplocha. "ScalaBLAST: a scalable implementation of BLAST for high-performance data-intensive bioinformatics analysis." *Parallel and Distributed Systems, IEEE Transactions on* 17.8 (2006): 740-749.

[29]ATLAS, Collaboration, et al. "ATLAS high-level trigger, data acquisition and controls technical design report." *ATLAS Technical Design Reports* (2003).

[30]Halbwachs, Nicholas, et al. "The synchronous data flow programming language LUSTRE." *Proceedings of the IEEE* 79.9 (1991): 1305-1320.

[31]Jefferson, David R. "Virtual time." *ACM Transactions on Programming Languages and Systems (TOPLAS)* 7.3 (1985): 404-425.

[32]http://www.pvfs.org/cvs/pvfs-2-8-branch-docs/doc/pvfs2-tuning/pvfs2-tuning.html

[33]Perumalla, Kalyan S. "μπ: a scalable and transparent system for simulating MPI programs." *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[34]Rodrigues, Arun F., et al. "The structural simulation toolkit." *ACM SIGMETRICS Performance Evaluation Review* 38.4 (2011): 37-42.

[35]Zheng, Gengbin, Gunavardhan Kakulapati, and Laxmikant V. Kalé. "Bigsim: A parallel simulator for performance prediction of extremely large parallel machines." *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004.

[36]Molina-Estolano, E., et al. "Building a parallel file system simulator." *Journal of Physics: Conference Series*. Vol. 180. No. 1. IOP Publishing, 2009.

[37]Scott, Joe. "Towards Building a Scalable Parallel Simulation of PVFS3."

[38]Liu, Yonggang, et al. "Towards simulation of parallel file system scheduling algorithms with PFSsim." *Proceedings of the 7th IEEE International Workshop on Storage Network Architectures and Parallel I/O (May 2011)*. 2011.

[39]Nguyen, Hai, and Amy Apon. "Hierarchical performance measurement and modeling of the linux file system." *ACM SIGSOFT Software Engineering Notes*. Vol. 36. No. 5. ACM, 2011.

Appendix

A. Contributions:

Sughosh Divanji(sdivanji@hawk.iit.edu):

1. Background study.
2. Develop first stage model in which client and server is on the same node.
3. Develop second stage model in which clients and servers are separated.
4. Debug the issues that were seen in development.
5. Evaluation of the simulation for I/O Throughput, Metadata Throughput, Comparison with FusionFS and NFS simulation and comparison with actual GPFS and PVFS deployments.

Raghav Kapoor (rkapoor7@hawk.iit.edu)

1. Background study.
2. Try to run Darshan workloads on the simulation.
3. Try to fix scaling issues by running torus network models on Fusion(fusion.cs.iit.edu) and Jarvis(jarvis.cs.iit.edu).
4. Debug the issues that were seen in development.

B. Challenges

The challenges that we faced in this project are:

1. We started with an assumption that file system models and storage system models already existed in the simulator. However, this turned out to be incorrect. So, we had to change the scope of the project from modeling burst buffers and data-aware scheduling to developing a parallel file system model from scratch.
2. CODES/ROSS is a project still under active development. So, the developers have not documented the APIs. Because of this we had to read the source code to understand the simulator. This made the learning curve a lot steeper.
3. Because CODES/ROSS is a project under development, there are bugs that exist in the simulator. Whenever, we found bugs we were blocked till they were fixed. We had to work with developers from ANL to report the bugs and provide them testcases to reproduce the bugs and get them fixed so that we could make progress on our work.

C. Acknowledgements

We would like to thank Ning Liu from SCS lab for helping us understand the simulator environment and previous work done in this area using CODES/ROSS. We would like to thank Jonathan Jenkins from Argonne National Lab for helping us to debug the issues we encountered and the quick turnaround on the bugs that we reported.