On the Design and Implementation of a Simulator for Parallel File System Research

Yonggang Liu, Renato Figueiredo
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL
{yonggang,renato}@acis.ufl.edu

Yiqi Xu, Ming Zhao
School of Computing and Information Sciences
Florida International University
Miami, FL
{yxu006,ming}@cis.fiu.edu

Abstract— **Due to the popularity and importance of Parallel File Systems (PFSs) in modern High Performance Computing (HPC) centers, PFS designs and I/O optimizations are active research topics. However, the research process is often time-consuming and faces cost and complexity challenges in deploying experiments in real HPC systems. This paper describes PFSsim, a trace-driven simulator of distributed storage systems that allows the evaluation of PFS designs, I/O schedulers, network structures, and workloads. PFSsim differentiates itself from related work in that it provides a powerful platform featuring a modular design with high flexibility in the modeling of subsystems including the network, clients, data servers and I/O schedulers. It does so by designing the simulator to capture abstractions found in common PFSs. PFSsim also exposes script-based interfaces for detailed configurations. Experiments and validation against real systems considering sub-modules and the entire simulator show that PFSsim is capable of simulating a representative PFS (PVFS2) and of modeling different I/O scheduler algorithms with good fidelity. In addition, the simulation speed is also shown to be acceptable.**

Keywords— simulation; parallel file system; I/O scheduling

I. INTRODUCTION

In modern High Performance Computing (HPC) systems, Parallel File Systems (PFSs) are widely adopted as the storage solutions, such as Lustre [1], PVFS2 [2], PanFS [3], GPFS [4] and Ceph [5]. To improve the system throughput or provide Quality of Service (QoS) guarantees, researchers also design I/O scheduling algorithms for these systems [6-10]. However, to thoroughly study those designs on large scale storage systems is costly. As a complement, simulation offers very useful insights in the performance trends and allows the pruning of the design space before implementation and evaluation on a real testbed or a deployed system.

In this paper, we present PFSsim (Parallel File System simulator), a PFS simulator allowing modeling and evaluations on PFS designs, network structures, I/O schedulers and workloads. Developed based on the OMNeT++ [11] network simulation framework, PFSsim provides these main features: *1) Pluggable scheduling module*: I/O scheduling algorithms can be easily deployed to any I/O buffer component. System information can also be probed by the scheduler. *2) High flexibility*: Common PFSs can be supported through configuration of its modules; detailed customization options are offered, including network topology, cache systems, and disk systems. *3) High fidelity*: Most of the major I/O performance impact factors in PFSsim are simulated and subsystems have been validated, allowing PFSsim to capture I/O performance with good accuracy.

The rest of the paper is organized as follows. Section II overviews the state of art on PFS simulators. Section III discusses real-world PFS-based storage systems, which provide the guidelines for the simulator design. Section IV describes the implementation details of PFSsim. Section V presents and evaluates the simulator validation results. Section VI concludes the paper and overviews future work.

II. RELATED WORK

Many distributed storage system simulators have been developed throughout the past decade. With different emphasis in design, each of them has outstanding features in particular areas. In this section, we briefly compare PFSsim with some of the most related works.

HECIOS [12] is a PFS simulator developed by the Parallel Architecture Research Laboratory for the tests on PVFS2. In HECIOS, detailed caching mechanisms, file systems and disk drives are simulated, and schedulers for the PFS server application and storage disk are introduced as well. The HECIOS simulator is capable of simulating very detailed PVFS2-based systems with good fidelity. But, as a tradeoff of offering such details, HECIOS sacrificed its flexibility in providing good support for more generic design problems. For example, it is not straightforward to tune the system to simulate parallel file systems not using BMI protocols, and for many modules on the I/O path (e.g., the network proxies), it is not easy to deploy specific scheduling algorithms on them.

SIMCAN [13] is a general-purpose storage network simulation platform, with the design goal of simulating a large variety of HPC systems with flexibility and scalability. The modular design facilitates users in system reconfiguration through tuning the detailed storage, memory, CPU and network subsystems. SIMCAN has modeled the computing nodes and the storage nodes, which are developed with detailed layered I/O modules mimicking the I/O stack in real systems. However, since SIMCAN is not specifically designed for PFS simulations, PFS features such as data stripping and metadata management are not supported. Moreover, I/O optimization researchers may also find it not straightforward to set up I/O schedulers in many subsystems of SIMCAN.

978-1-4799-0218-7/13/$31.00 ©2013 IEEE

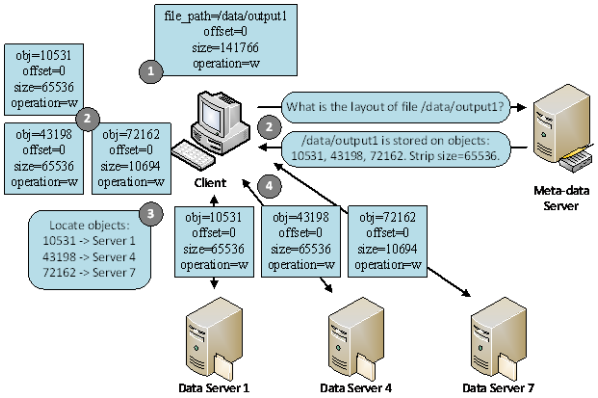


Fig. 1. The process of an example user I/O request in PFS

The CODES [14] simulator is proposed for the evaluation of exascale storage system designs. CODES models the Argonne Leadership Computing Facility's (ALCF) computing and data storage environment. Several hardware and software parameters, such as the network throughput and latency, CIOD transfer size, and data stripe size are configurable. For the simplified architecture and ROSS simulation platform it is built on, CODES is able to simulate systems at a large scale. However, as some of the I/O subsystems are omitted in CODES (such as network devices and caching subsystems), it may lose simulation fidelity in many use cases.

IMPIOUS [15] is a simulator designed for fast PFS design evaluations. IMPIOUS simulates a simplified PFS architecture by providing core components: the clients, the Object Storage Devices (OSDs) and a simplified network. The limitation of this work is that it has very limited capability of simulating and customizing the details of a storage system, so many aspects of the real systems cannot be reflected in the simulation result.

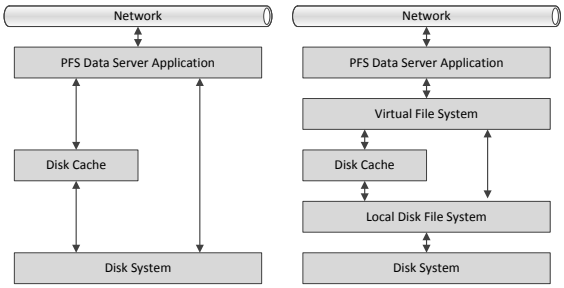
PFSsim differentiates from the aforementioned work in that it models all I/O modules in common PFSs to provide a flexible, complete and detailed PFS testbed for comprehensive experiments. PFSsim also fully supports scheduler deployments, in that it provides pluggable scheduler modules, as well as APIs for retrieving the system runtime information and collaborating with other schedulers.

III. System Modeling

To achieve the goal of simulating a variety of PFSs and I/O schedulers, we studied the system architecture and I/O overhead in many existing systems. In this section, we discuss our approach in modeling the PFS system and I/O schedulers.

A. Parallel File System Overview

In parallel file systems, data are often managed in terms of "objects" (or equivalent terms). A typical PFS has three major components: *1) Client*: Provides the interface (e.g., POSIX) for the user processes to access the PFS, and requests data I/O by communicating with data servers and metadata servers. *2) Data Server (or Object Storage Devices, OSDs)*: Serves data to clients. Data are often stored in the form of objects on the local storage system. *3) Metadata server*: Manages the mapping from PFS file name space to PFS storage object name space, and handles file metadata operation requests.



(a) PFS based on physical disk (b) PFS based on native FS

Fig. 2. The modules and I/O flows in the data server

During the processing of a normal user I/O request (data read/write operation), the above three PFS components are involved in four major steps (Fig. 1). *1) File I/O request*: The user process initializes the I/O by sending a request to the PFS client application. *2) Object mapping*: The PFS client application finds out how the target file is stored in PFS, querying the metadata server if required. Based on the computation of data striping, the client knows the data are stored in some objects. *3) Object Locating*: The client locates the objects on the data servers storing them. This mapping information is often locally available for the client. *4) Data transmission*: The client initiates the data transmission with the corresponding data servers. The data I/O lasts until all the data are transmitted. Note that the I/O process shown in this example is a general data processing approach, and different PFS protocols may implement them in slightly different ways.

B. The PFS Subsystem

In a PFS subsystem, there are four aspects that may have significant impacts on system I/O performance [16]. *1) Metadata management*: Metadata I/O is often a critical part of system data I/O. In PFS, one or more metadata servers can be utilized. In the latter case, metadata servers may be organized in a pattern, for example, the dynamic metadata management in Ceph. *2) Data placement strategy*: A common data placement technique is to stripe data into data objects, which are distributed onto multiple data servers. The goal of data distribution schemes is to statistically balance the load on each data server, but the implementations may vary among PFSs. For instance, PVFS2 delegates to users the responsibility to specify the stripe size and data servers to store specific data files while many other PFSs do not. *3) Data replication model*: Duplicated data may incur overheads in duplicate I/O and data integrity checks. For example, in Ceph if data replication is enabled, every write operation will be committed to multiple OSDs in a placement group, which brings additional overhead. *4) Data caching policy:* Data caching on the data server or the client may improve the PFS I/O performance, at the cost of the data copy coherency management. PanFS implements data caching at both client and server. Some PFSs, such as PVFS2, do not support client caching by default, but note that the native file system may still have caching enabled.

C. The Data Server Subsystem

Typically, there are two types of I/O flow architectures inside a PFSsim data server (Fig. 2). Fig. 2 (a) shows the PFSs that are built upon the physical disks, in which the data server applications manage the disk cache and block devices directly.

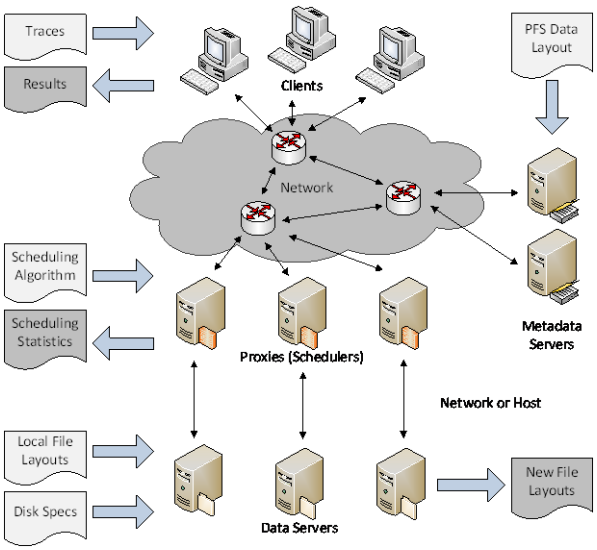


Fig. 3. The simulated architecture of an example PFS server

These PFSs include Lustre, GPFS and PanFS. Fig. 2 (b) shows the PFSs that are built on the native file systems. These PFSs often interact with the local storage system through the virtual file system interface. PVFS2 and Ceph are two representative PFSs designed in this scheme.

D. The I/O Scheduler Subsystem

In recent years, many distributed storage system I/O management solutions have been proposed, involving both centralized and decentralized schemes, as well as various deployment locations for different system needs. It is common for I/O schedulers to be transparently deployed on network nodes such as gateways or proxies. In Façade [6], the I/O scheduler is deployed on a *centralized proxy*, which interposes all the system I/O and forwards them to the storage devices. In Xu et al. [7], the I/O management policies are deployed on *per-server proxies*, which intercept I/O and virtualize the data servers to the system clients. In Wang et al. [8], the scheduling algorithms are deployed on distributed *Coordinators* [9]. Meanwhile, Scheduling algorithms can also be implemented on the PFS data servers as part of the fundamental PFS services. Some PFS have internal scheduling modules on the data server applications. For instance, Ross et al. [10] describe the server-side scheduling mechanisms in PVFS2 data servers. Specific scheduling algorithms also exist on the lower layers of a data server, for example, the disk systems may adopt the SSTF or SCAN algorithms to reduce the total disk seek time.

IV. Simulator Implementation

PFSsim is implemented based on the network simulation framework OMNeT++. The code is open source and available to the community [17]. PFSsim provides the following major components: PFS client, PFS metadata server, PFS server and network devices. Those components are connected by the simulated network, and the I/O scheduler can be deployed inside any component. Fig. 3 illustrates the simulated architecture of an example PFS with per-server proxies [7], including interconnection network clients, metadata servers and I/O proxies. The proxies and data servers are linked by another network (e.g., SAN) or located on the same host machine on a "one proxy per server" basis. The I/O schedulers are deployed on the proxies. The major modules of PFSsim are described in detail in the following subsections.

A. PFS Client

The PFS Client contains two major modules: the user application module and the PFS client application module. These modules communicate by passing file I/O requests and responses. A user application module reads the I/O traces from a trace file, and outputs each data I/O's processing timestamps to the result file. Each trace file represents the workload from one application, and it contains the information for every data I/O, including start time, file ID, offset, size, R/W, and application ID. This trace file can be generated by a real system at the I/O clients. The PFS client application module may query the metadata server for the data layout of PFS files, and it conducts the data I/O with the data servers. The file metadata and I/O data are cacheable at the PFS client application module.

B. PFS Data Server

Using a data server with Linux kernel 2.6 as prototype reference, we designed the internal modules of the data server in PFSsim. The architecture is the same as the one shown in Fig. 2. The *PFS data server application* module receives the PFS I/O requests from the clients and maps the objects to the files in the native file system or blocks on the local disk. It may buffer the traffic to schedule the I/O. The *virtual file system* module translates the requests with offset and size in the target files to the page address space. It issues I/O to the disk cache or the disk file system (with *O_DIRECT* set). In the *disk cache* module, the cache size, page size, and cache read/write speeds are configurable. The disk cache has realized a page table by a per-file ordered doubly-linked list (*page-table-list*) with each node storing a chunk of adjacent pages that have identical page flag settings (*e.g.*, *PG_dirty*). Each node in the page-table-list is also in a page reclaiming list, where the LRU (Least Recently Used) algorithm is implemented. The *local disk file system* module manages the block layouts for the files in the generic block address space exposed by the disk system. It is able to read the file layout from an input script and mimic the EXT2 disk space allocation schemes. In the *disk system* module, the disk I/O speed is modeled based on statistics extracted from real disks. In PFSsim, disk I/O speed is modeled based on the disk tracks, I/O size, R/W and the disk head movement distance. The simulator package includes scripts for users to easily extract the disk parameters from a real disk device.

C. PFS Metadata Server

The metadata servers provide the data layout and other metadata to the clients. The data layout information is specified by the scripts from users. In the current version, all the PFS metadata are stored on the metadata server. Deployment of multiple metadata servers is supported in PFSsim. Cooperative metadata management schemes are also possible.

D. I/O Scheduler

In PFSsim, the I/O scheduler module is capable of: *1)* Offering flexibility for users to easily associate I/O scheduling algorithms to any I/O buffer in the system. PFSsim couples the scheduling algorithms with the I/O queue classes, as shown in

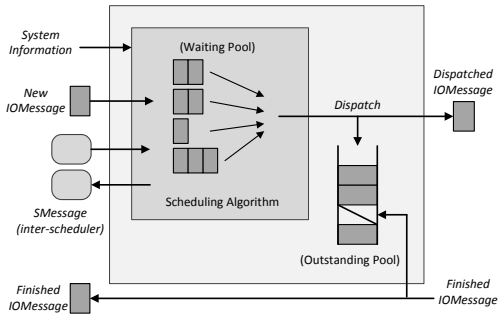


Fig. 4. The structure of the I/O queue class

Fig. 4. The I/O queue is widely implemented in PFSsim for the simulation of I/O buffers. *2)* Enabling inter-scheduler information delivery interface for decentralized scheduling. The *SMessage* objects are used to carry the scheduling information between schedulers (Fig. 4). These messages can be sent through the existing network connections, or through separate channels defined by users. *3)* Providing ways for the I/O schedulers to probe the real-time system status. If the status is locally available to the scheduler, the user can probe the system status by accessing the module's status variables. Otherwise, the module providing the status can encapsulate the information in *SMessages* and send them to the schedulers.

E. Network and Network Devices

The *network connection* is modeled by the OMNeT++ components, which offer bandwidth and latency configurations. The *router/switch* module redirects the packets in the network. The packets may be buffered and delayed on the *router/switch* due to data congestion. The *proxy* module is introduced to host the network I/O schedulers. The I/O queue class can be implemented on the proxy modules so that the I/O scheduling algorithms can be deployed.

V. VALIDATION AND EVALUATION

The testbed physical cluster contains 16 nodes. Each node contains 2 six-core 2.4GHz Opteron CPUs, 24GB of RAM and 7200 RPM SAS disk. Between any two nodes, the average network bandwidth is 970Mbit/sec, and the average network latency is 0.075ms. The local loopback network has 0.01ms latency, and 11.3Gbit/sec bandwidth. PVFS2 with Linux kernel 2.6.32 is deployed on each node. The local disk file systems are in EXT3 format. In each Linux kernel, the *dirty_ratio* is set to be 20%. PFSsim is configured according to the parameters gathered from the real system, including disk system, caching system, PFS, and network. We used IOR [18] as the benchmark workload generator. The I/O traces for PFSsim are generated on the clients. All tests are run 5 times and the average values are used in the result analysis.

To validate the model of a data server, we have implemented a single-server PVFS2 system. The client is located on the same node as the data server. The total I/O size is 1GB, and the I/O is directed to one file. Fig. 5 shows the average throughput of the real and simulated systems in three test cases: reading from cache, reading from disk and writing. We notice that because of the metadata overhead, smaller request sizes result in lower throughput. But the throughput eventually reaches a ceiling when request size is large enough;

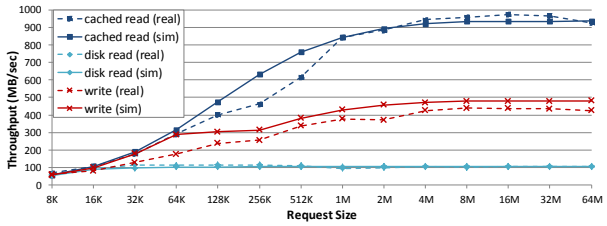


Fig. 5. Average throughput of local single-server PFS read/write

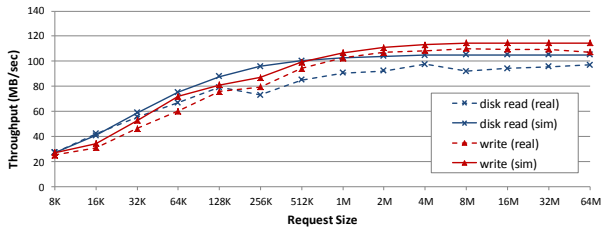


Fig. 6. Average throughput of remote single-server PFS read/write

this is because the cache or disk I/O is saturated. To validate the network, we have run another two sets of tests with the client deployed on another machine. Fig. 6 shows the average I/O throughput for reading from disk and writing. In these tests, the lower network bandwidth brings significant overhead. In this sequence of tests, the simulation results show good fidelity. The discrepancies in those simulated results are mainly due to the network and PFS-specific protocols that are abstracted or omitted in PFSsim.

In the PFS validation, we have deployed PVFS2 data servers across 8 physical nodes, using one of them also as the metadata server. The data files are evenly distributed onto the data servers with stripe size 64KB. Each client sequentially accesses a 512MB file, and each I/O request is 256KB. Tests of sequential disk reads and sequential writes are conducted, with the number of clients varying from 1 to 128. Fig. 7 shows the total system throughput for the read/write I/O on both the real system and PFSsim. The simulation results follow the trend of real system results well. For the read I/O tests, the system throughput grows as the number of clients grows when clients are fewer than 16, which is due to higher server utilization. But the throughput decreases afterwards. This is because the disk head movement is increased for seeking among more file locations while serving different I/Os concurrently. For the write I/O, the total I/O throughput grows due to higher server utilization, but the throughput is greatly downgraded with 128 clients. This is because the dirty pages on each data server (in total 8GB) exceed the *dirty_ratio* threshold (about 4.8GB). Therefore, more overhead is brought by explicit page write-backs.

We evaluate the I/O scheduler of PFSsim on proxies. The testbed contains 4 PVFS2 data servers and 1 metadata server. On each data server node, a proxy is also implemented to intercept the I/O traffic. We deployed the *SFQ(4)* algorithm [19] on each of the proxies. We divide the clients into 2 groups (G1 and G2), each with 16 clients. Each client issues sequential read/write I/O to a 400MB file; each I/O request is 1MB. All files are evenly distributed on all the data servers with stripe

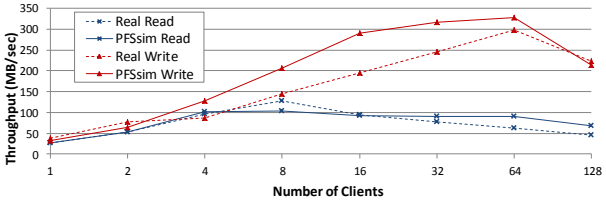


Fig. 7. Average throughput of PFS read/write I/O

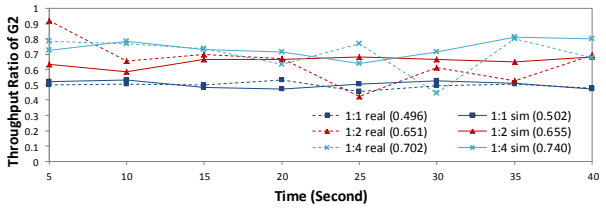


Fig. 8. Throughput ratio of G2 in the SFQ(D) algorithm

size 4KB. Three sets of tests are conducted, with weight ratios 1:1, 1:2, 1:4 for G1:G2 in SFQ(4). We monitor the real-time system throughput ratio change and the average I/O throughput ratio during the first 40 seconds of system runtime. Fig. 8 shows the changes in throughput ratio of Group2 during the runtime. The average throughput ratios are in the brackets in the legend. The simulated average throughput ratios are shown to have good accuracy. Also, the simulated results are able to show that the oscillations in the I/O throughput grow as the share ratio becomes more imbalanced.

Table I shows the simulation efficiency of PFSsim. We run tests on a personal laptop with quad-core 2.13GHz Intel Core i3 CPU and 3GB of RAM. In total, each client issues 128MB sequential read/write data I/O to a file, and the total simulation time is measured with various request sizes.

TABLE I. SIMULATION TIME SPAN

Request Size	Read			Write		
	8:8[a]	*512:8*	*512:32*	*8:8*	*512:8*	*512:32*
16KB	9s	2095s	1909s	9s	2832s	1142s
256KB	8s	1618s	1220s	7s	1751s	548s
4MB	7s	1318s	1404s	5s	1578s	523s

[a] The notation in this row stands for number of clients vs. number of servers.

VI. CONCLUSION AND FUTURE WORK

This paper shares the experience in building a general purpose PFS simulator that is easy to use, modular, and flexible to support customizations of different parallel file system design points. The approach in this paper presents a storage system modeling methodology, and the PFSsim simulator can be delivered as a fundamental I/O system simulation tool for the PFS research community. In the future work, PFSsim will be validated with more real workloads and more PFS systems. Large scale parallel simulations will also be researched.

REFERENCES

[1] Sun Microsystems, Inc., "Lustre file system: high-performance storage architecture and scalable cluster file system", Sun Microsystems, Inc., Santa Clara, CA, white paper, 2008.

[2] P. Carns, W. Ligon, R. Ross, and R. Thakur, "PVFS: A parallel file system for Linux clusters", in Proc. the 4th annual Linux Showcase & Conference, Atlanta, GA, 2000, pp. 317-327.

[3] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas activeScale storage cluster-delivering scalable high bandwidth storage", in Proc. the 2004 ACM/IEEE Conference on Supercomputing (SC'04), Pittsburgh, PA, 2004, p. 53.

[4] F. chmuck and R. Haskin. "GPFS: A shared-diskfi le system for large computing clusters", in Proceedings of the 1st USENIX Conference on File and Storage Technologies, pp. 231–244, Monterey, CA, Jan. 2002.

[5] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distrib-uted file system", in Proc. the 7th symposium on Operating Systems Design and Implementation (OSDI'06), Seattle, WA, 2006, pp. 307-320.

[6] C. R. Lumb, A. Merchant, and G. A. Alvarez, "Façade: virtual storage devices with performance guarantees", in Proc. the 2nd USENIX Conference on File and Storage Technologies (FAST'03), San Francisco, CA, Mar. 2003, pp. 131-144.

[7] Y. Xu, L. Wang, D. Arteaga, M. Zhao, Y. Liu, and R. Figueiredo, "Virtualization-based Bandwidth Manage-ment for Parallel Storage Systems". in 5th Petascale Data Storage Workshop (PDSW'10), New Orleans, LA, Nov. 2010, pp. 1-5.

[8] Y. Wang, and A. Merchant, "Proportional-share Scheduling for Distributed Storage Systems", in Proc. the 5th USENIX Conference on File and Storage Technologies (FAST'07), San Jose, CA, Feb. 2007, pp. 47–60.

[9] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. "Fab: Building distributed enterprise disk arrays from commodity components", in Proc. the 11th international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Boston, MA, Oct. 2004, pp. 48-58.

[10] R. B. Ross and W. B. L. Iii, "Server-Side Scheduling in Cluster Parallel I/O Systems," Calculateurs Parallèles Journal, Special Issue on Parallel I/O for Cluster Computing, 2001.

[11] A. Varga, "The OMNeT++ discrete event simulation system", in Proc. the European Simulation Multi-conference (ESM'01), Prague, Czech Republic, Jun. 2001.

[12] HECIOS, 2009. Available: http://www.parl.clemson.edu/hecios/

[13] A. Núñez, J. Fernández, J. D. Garcia, L. Prada, and J. Carretero, "SIMCAN: a SIMulator framework for Computer Architectures and storage Networks", in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops* (Simutools '08), Brussels, Belgium, Mar. 2008.

[14] N. Liu, C. Carothers, J. Cope, P. Carns, R. Ross, A. Crume, and C. Maltzahn, "Modeling a leadership-scale storage system", in Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics, Torun, Poland, Sep. 2011.

[15] E. Molina-Estolano, C. Maltzahn, J. Bent, and S. A. Brandt, "Building a parallel file system simulator", poster session presented in SciDAC'09, San Diego, CA, Jun. 2009.

[16] Y. Liu, R. Figueiredo, D. Clavijo, Y. Xu, and M. Zhao, "Towards simulation of parallel file system scheduling algorithms with PFSsim", in Proceedings of the 7th IEEE International Workshop on Storage Network Architectures and Parallel I/O, Denvor, CO, May 2011.

[17] PFSsim, 2013. Available: http://www.github.com/myidpt/PFSsim

[18] Interleaved or Random (IOR) Benchmark, Available: http://www.cs.sandia.gov/Scalable_IO/ior.html

[19] W. Jin, J. S. Chase, and J. Kaur, "Interposed proportional sharing for a storage service utility", in Proc. the joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'04), New York, NY, Jun. 2004, pp. 37-48.