HPIS3: Towards a High-Performance Simulator for Hybrid Parallel I/O and Storage Systems

Bo Feng, Ning Liu, Shuibing He, Xian-He Sun
Department of Computer Science
Illinois Institute of Technology, Chicago, IL
Email: {bfeng5, nliu8}@hawk.iit.edu, {she11, sun}@iit.edu

Abstract—**The performance gap between processor and storage device has continuously increased during the past few decades. The gap is further exacerbated recently because applications are becoming more data-intensive in both industry and academia. Traditional storage devices, such as hard disk drives (HDD), fail to keep up with the paces of this growth. A known solution is to use solid state drives (SSD) as fast storage. Due to high cost of SSD, data and supercomputing centers usually adopt a hybrid storage system, which consists of a combination of HDD and SSD I/O servers. However, hybrid I/O and storage systems have increased the complexity, making SSD often underutilized. The configuration and utilization of HDD/SSD hybrid systems is a lasting phenomenon. In this study, we propose a high performance hybrid parallel I/O and storage simulator, HPIS3. As a co-design tool, HPIS3 is capable of simulating a variety of parallel storage systems, especially under hybrid scenarios. The experimental results show that the lowest error rate is 2%, and the average is 11.98%.**

Keywords*—*HPC; Parallel I/O; Simulation; SSD

I. Introduction

High performance parallel I/O and storage systems are designed to address the I/O requirements in HPC systems. Parallel File Systems (PFSs), such as PVFS[1] [1], Lustre [2], GPFS [3], PanFS [4] and Ceph [5], are the primary solutions in the HPC domain. In these file systems, a large file request typically is divided into multiple small sub-requests with a fixed stripe size. However, intensive small accesses on HDDs can degrade the performance severely. This degradation can be alleviated by using larger file stripe size, which, on the other hand, loses I/O access parallelism and hence the aggregated I/O bandwidth. Optimizing parallel file systems is an uneasy task.

Solid state drives (SSD) provide excellent properties beyond that of HDDs, including well improved small random requests performance [6] and low access latency. As SSDs are becoming more reliable, they start to be deployed in enterprise and high performance computing facilities [7]. However, their deployment is still limited by capacity and cost constraint. It is unlikely that traditional spinning hard drives will be fully replaced by SSDs in the near future. An alternative is to use a heterogeneous storage system consisting of both HDD and SSD. Heterogeneity is useful in HPC because SSDs can provide an opportunity to reduce I/O latency and increase IOPS, and conventional HDDs can play an important role in supporting a high degree of parallelism and providing sufficient capacity within limited budget.

The task of exploring complex heterogeneous system is challenging in current system environment. Hybrid parallel I/O and storage systems have a variety choices of hardware and software configurations and an even richer set of optimization algorithms. It is almost impossible to obtain an optimal system configuration without the help of an efficient co-design tool. For example, SSD and HDD have different I/O latencies, and applications may have different access patterns and data layout schemes, which makes the PFS tuning costly. In order to test and boost parallel I/O and storage systems, this paper presents a Hybrid Parallel I/O and Storage System Simulator (HPIS3), a co-design tool targeting the optimization of hybrid parallel I/O and storage systems. HPIS3 focuses on HDD/SSD hybrid parallel systems, where a set of SSDs and HDDs are deployed as storage nodes.

The major contribution of this paper is the design and implementation of HPIS3. We demonstrate the validity of HPIS3 through performance comparison between the simulation and real system. We further demonstrate the capability of HPIS3 through a use case design and experiment. In addition, HPIS3 features two distinct types of hybrid system design, namely buffered-SSD and tiered-SSD storage systems. Experimental results show that HPIS3 can scale up to 256 MPI processes.

II. Related Work

A. Storage Device and Parallel I/O Simulators

DiskSim and its SSD extension are well accepted simulators for single disk behaviors. However, they cannot simulate the HPC environment directly. Substantial researches have focused on the performance modeling under parallel I/O and storage systems. For example, Liu et al. [8] built a simulation system for the Blue Gene/P supercomputer in Argonne Lab, but this work was done under homogeneous storage systems. Narayanan et al. from Microsoft analyzed the tradeoffs of migrating the storage from HDDs to SSDs [9]. PFSsim [10] is another trace-driven simulator, which can simulate the PFS design on distributed storage systems. It includes I/O schedulers, network structures, and workloads. Researchers from UCSC systems research lab and Los Alamos National Lab built a parallel file system simulator, focusing on storage cost and power [11].

B. SSD Usage in I/O and Storage Systems

Using SSDs as a cache for traditional HDDs is a widely adopted strategy in I/O system design. FlashCache [12] uses flash-based memory to achieve high performance as well as

[1] Now this project is renamed to and maintained by OrangeFS.

low energy consumption. SieveStore [13] traces the most popular blocks and places them onto SSD. iBridge [14] forwards the unaligned sub-requests to SSD to achieve the most cost-effective improvement. Many efforts have been made to enlarge SSDs advantages and avoid their disadvantages [6], [15]. For example, to improve the performance of data intensive applications, San Diego Supercomputer Center has built a large SSD-based high-performance computing cluster, called Gordon [7].

C. *PFS Performance Tuning*

System-wide performance tuning is a prevalent topic in computer science. A variety of works have focused in this area. Anderson et al. [16] built a system, which automates the design and configuration process for the storage system. It is facilitated with the interactive design loop to optimize the local disk array configuration. When the local storage array becomes relatively large scale, Minerva [17] uses declarative specifications of applications requirements, device capabilities, and optimizes to explore the search space for possible solutions. Behzad et al. [18] proposed a work of auto-tuning for optimizing I/O performance by using HDF5 I/O middleware. CALCiom [19] is a framework to mitigate the I/O interference through dynamic selection of scheduling policies.

III. The Parallel I/O and Storage Device Simulation Model under PVFS

HPIS3 is built on Rensselaer Optimistic Simulation System (ROSS) [20], a parallel simulation platform. In this section, we first introduce ROSS and then present the design and implementation of the PVFS models and the storage device models.

A. *ROSS Platform*

ROSS is developed in ANSI C, and features time-warp optimistic algorithm and reverse computation for fast and efficient parallel event processing. ROSS is based on logical process (LP) models and provides a set of APIs for building parallel discrete event models. LPs are abstractions of simulated physical processes. They act like real processes in the system and are synchronized by Time Warp protocol [20]. Many successful systems leverage the functionalities provided by ROSS [21], [22], e.g CODES [23], BigSim [24] etc. In this study, we implement HPIS3 on top of ROSS platform to achieve best simulation efficiency and system scalability.

B. *Simulation Architecture Overview*

HPIS3 models the parallel I/O and storage system in HPC environment. Figure 1 presents the overview of its architecture. HPIS3 has 4 main components: 1) application workloads; 2) PVFS clients; 3) PVFS servers and 4) storage devices. In this design, SSDs are used both in the buffer layer and the storage layer in terms of configurations. This flexibility brings optimization space for PFS performance tuning. Applications access patterns can vary at runtime, and the data layout on PFS can also impact the overall performance. The central idea in this design is that we only focus on simulating the small sub-requests, which are the bottleneck along the data path and avoid modeling other details to keep the HPIS3 simple and light-weight.

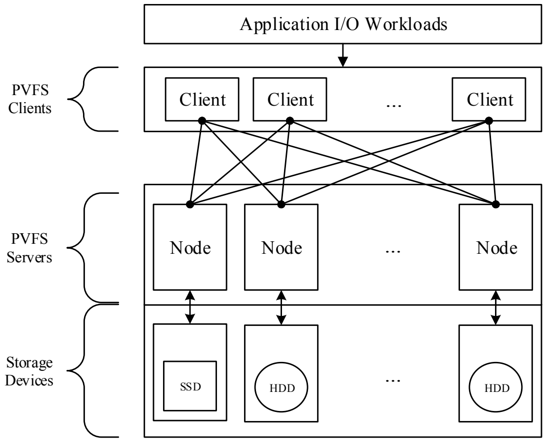

Fig. 1: Simulator Architecture

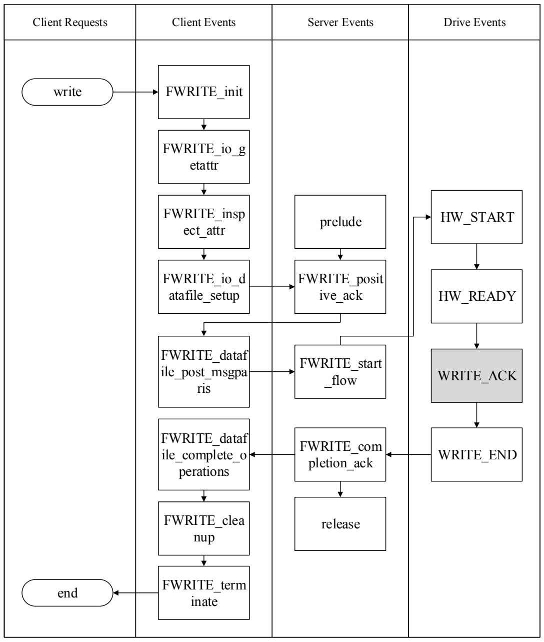

Fig. 2: Write Event Flow Chart in Simulator

The application clients are driven by the actual application trace. The traces are collected by using IOSIG [25]. IOSIG is a fully functional MPI/POSIX based tracing tool, which is capable of capturing the applications I/O logic order with accurate timestamps. HPIS3 reads trace files and generates events to trigger client behaviors. In Fig. 1 PVFS clients and servers are modeled as different logical processes on separated layers. We use two types of LPs to model HDD and SSD, and they are connected onto the server node LPs. All the communication latency is configurable by an input file. By given different network latency parameters, the physical network topology between PVFS clients and PVFS servers can be simulated.

In HPIS3, all the communication between LPs are via simulated message passing on ROSS platform. Figure 2 illustrates the detailed event workflow of how a write request is handled by both the PVFS clients and servers. First, HPIS3 generates file request events, which triggers the simulated requests to be dispatched from the appropriate clients. Then the requests are divided into sub-requests, which are in turn sent to specific servers. The server hands over all the sub-requests to the hardware model, where the simulated requests are handled. The following sections will specifically elaborate the modeling of file requests, queues, PVFS clients, PVFS servers, and two types of storage hardware: HDD and SSD.

C. File Requests and Request Queue Modeling

File requests in HPIS3 are simulated by wrapped messages. Messages in HPIS3 are customized by requirements of each LP model. To simulate file requests, we wrap three typical properties: *file_id*, *length* and *file_offset*. These messages are handled by Client LPs, Server LPs and Drive LPs. Message types are predefined to identify the type of messages. All LPs handle the request according to both the type of request and their states. In order to retrieve the statistics, we also add some properties to requests such as sender id, receiver id, start time and end time. In HPIS3, we simplify the calculation of the network latency for each request to a variable, which is a configurable parameter in the startup configuration file.

Request queues in HPIS3 are simulated by some special variables in the state of LPs. ROSS handles messages in a first-come-first-serve manner. A later request could be processed sooner. To keep the logical accuracy, for example, if the request queue is defined as first-come-first-serve without preemption, one variable will be added to the state of a LP. This variable records the next available time of this LP to handle a request. So the logical time of handling requests can be calculated and also the logical request waiting time can be calculated in this manner. If a multi-queue is required, an array of variables will be added to the LP.

D. PVFS Client Modeling

In Figure 1, PVFS Clients are the clients inside the users' applications. In this simulation, they are used to issue all the requests from applications. The striping mechanism is also implemented. There is a fixed stripe size defined. This mechanism simulates the real design of some parallel file systems. HPIS3 triggers all the requests from the trace files to the corresponding clients (ranks). For each PFS request, ranks can monitor the status of each request and sub-request and stripe the request into multiple sub-requests. A request is finished when all its sub-requests are finished. The dependencies of requests are maintained by a linked list, meaning that the request on the top of the list has the highest priority. Once the file server node receives a sub-request, the file node puts it into the tail of the input pipeline and continues on executing the head of the pipeline. When a sub-request is finished by the file node, the feedback (or data) will be put into the output pipeline, which will then be sent to the corresponding clients. By giving a specific routing time penalty of all the messages, the finish time of sub-requests, and the finish time of the whole request can be counted. This is to simulate the total access time for a request is determined by the slowest sub-request. This mechanism is exactly the same as those in PFSs such as PVFS.

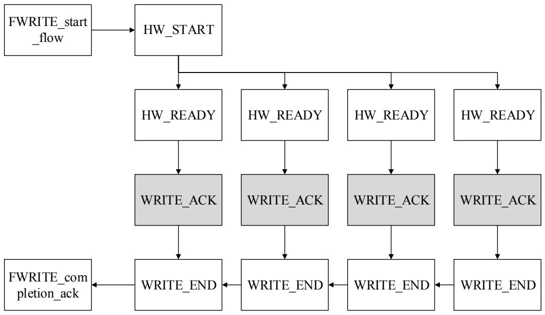

Fig. 3: Concurrent Write Requests on SSD

Any request from the applications might be striped into multiple sub-requests. For example, three cases occur in PVFS, if there are four file server nodes. The sub-requests will be dispatched in a round-robin fashion. The three cases where a file request involves a different number of file servers are: (a) The request is smaller than the stripe size (which is a fixed size in PVFS), and hence only one file server is involved. The request itself will be dispatched directly to the appropriate server. (b) While the request spans across more than three stripe sizes, three file servers are involved. The request will be stripped into three sub-requests, the first two of which have the same stripe size and the last one has the remaining size, which could be less than a single stripe size. (c) When the size of the request is larger than the sum of the stripe size on the four servers, according to the round-robin algorithm, all the four file servers are involved. The request will then be divided into five sub-requests. The first four sub-requests will be dispatched to each of the servers, and the last sub-request will be sent to the first server in a round-robin fashion.

E. PVFS Server Modeling

Each PVFS Server works as a standalone server. PVFS is derived from the state machine programming pattern [1]. Fig. 2 shows all the states are used from both the client side view and the server side view in a write progress. In the simulation model of PVFS server, multiple critical states are simulated, such as start positive actions and open the response flow. PVFS server LPs handle the events at their appropriate states. Server LPs can handle events separately as they can work in parallel. For example, if there are four server nodes, each responds to the requests from any client in MPI-IO without intervening others. There are two queues in each of the server nodes: 1) the first queue is used to buffer all the requests from the client side, which will then be sent to the hardware model; 2) the second queue stores the response from the hardware, which will be then sent to the clients. At the state of opening flow, server LPs send messages to HDD/SDD LPs; at the state of completion, they receive messages back from HDD/SSD LPs.

F. Storage Device Modeling

Storage devices are fundamentally different in hardware structures, which affect I/O performance. We build the models for two types of storage devices: HDDs and SSDs. HDDs, as conventional storage devices, is simulated in a light-weight model. Flash based SSDs have more channels to handle current requests in simulation. We simulate this feature by

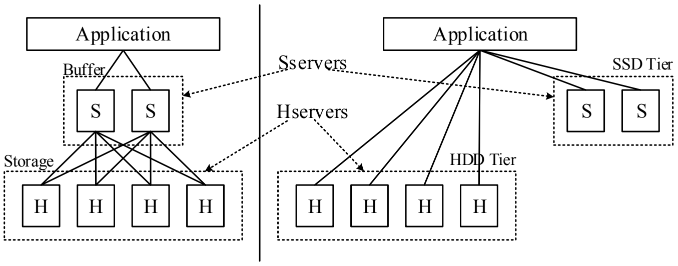

Fig. 4: Buffered-SSD and Tiered-SSD Architectures

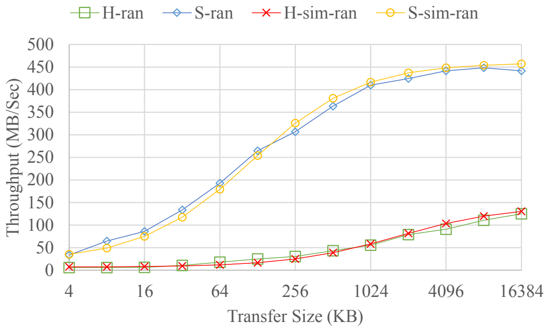

Fig. 5: Random Write Performance Comparison between Real System and HPIS3

providing more queues for the model to simultaneously handle a fixed number of current requests. Figures 2 and 3 give some examples on how file write requests are handled from the clients to the servers and storage devices.

1) HDD: For hard disk drives, all data is recorded by magnetizing a thin film of ferromagnetic material. There is one head for each magnetic platter surface on the spindle, and an arm moving the heads across the platters. To simulate these properties of HDD, our model records disk states by some variables such as number of open files, last request offset. We trace the states of the simulated disks and each request access onto the disk, and hence the disk model knows whether the request is sequential or random. Thus, the overall performance for HDD could be aggregately calculated.

2) SSD: SSDs have no mechanical moving parts, and are less sensitive to random data accesses. Thus they can provide lower latency compared to HDDs. A typical hardware structure of SSDs has several critical components: Die, Package, Channel, and FTL. A single flash chip is called a “Die”. A package represents a group of dies. Multiple dies within a package are connected to the same Channel in the bus. Figure 3 shows an example of requests from a PVFS server to SSD, in which multiple channels are simulated. In HPIS3, we add latency penalties for all the components and performance benefits from multi-channels.

G. Hybrid PVFS I/O and Storage Modeling

We refer to the term ‘hybrid’ as the layout of two different types of storage nodes. In our case, the storage nodes can consist of either HDDs or SSDs. HPIS3 features two distinct types of hybrid system design, namely buffered-SSD storage system and tiered-SSD storage system. To simulate a Buffered-SSD storage system, all SSD based server nodes are responsible for all read/write requests in a LRU algorithm; while in Tiered-SSD storage system, all requests are striped onto each server node accordingly. We define a set of server nodes equipped with HDD as Hservers and that with SSD as Sservers.

1) Buffered-SSD Storage System: In this configuration, a set of Sservers is responsible for acting as a buffer layer in front of Hservers. This configuration is featured from our previous work, such as burst buffer from Liu et al. [8] and S4D-Cache from He et al. [26]. However, the buffered-SSD is slightly different from them. As illustrated in Fig. 4, all write requests are routed to Sservers before Hservers if the following conditions can be filled: a. write to Sservers if the capacity of Sservers is allowed immediately; b. write to Hservers if the capacity of Sservers is not allowed immediately; c. to evict old requests, all data on Sservers will be forwarded to Hservers in an asynchronous manner.

2) Tiered-SSD Storage System: This configuration combines a set of Hservers with Sservers together to support I/O requests accordingly. This style is also featured from our previous work CARL [27]. In this configuration, a file is logically divided into multiple regions. Some regions are placed on Hservers and some regions on Sservers. All request are routed to the predefined region placement. If the range of a request involves more than one region, the request will be further striped into smaller sub-requests according to the edges of the regions.

IV. Experiments

This section shows the experiments which serve as verification for HPIS3 on multiple nodes in the hybrid file systems. Then, the performance of the simulator is studied by increasing MPI processes. Finally, a performance optimization case is shown from tuning the simulation system.

A. Experimental Setup

Our test bed is a 65-node SUN Fire Linux cluster, namely HEC. Each node has two AMD Opteron processors, 8GB memory and a 250GB SEAGATE HDD. The SSD cards are PCI-E X4 100GB SSD (OCZ-REVODRIVE X2). The MPI version is MPICH 3.0.4 stable release and the parallel file system is OrangeFS version 2.8.6. IOR [28] and its traces are used to benchmark the whole system with varying configurations. IOR is a benchmark with synthetic features that are good at mimicking many kinds of application behaviors, such as sequential read/write and random read/write. The traces are from logging the I/O signatures of running IOR by IOSIG [25]. With MPI-IO built with the PVFS file system, the IOR can vary behaviors in multiple ways, and hence mimic the HPC applications. HPIS3 is driven by the trace.

B. Simulation Validity

HPIS3 is configured through a set of parameters upon initialization. The simulation accuracy can be adjusted according to the experimental results from real systems. We run experiments of IOR random writes on real PVFS system and on HPIS3 respectively. Figure 5 illustrates the performance comparison. The experiments are carried out by using 8 clients, 4 HDD-servers and 4 SSD-servers respectively. *H-ran* and *S-ran* are real results for HDD- and SSD-servers. *H-sim-ran* and

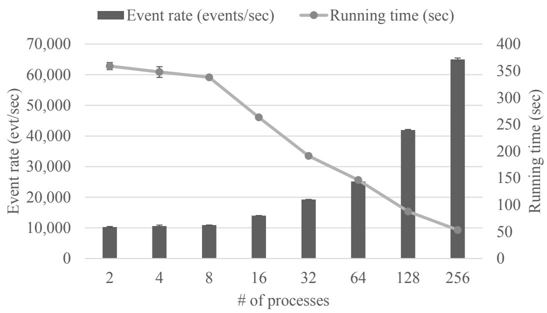

Fig. 6: HPIS3 Performance as a Function of Number of Processes

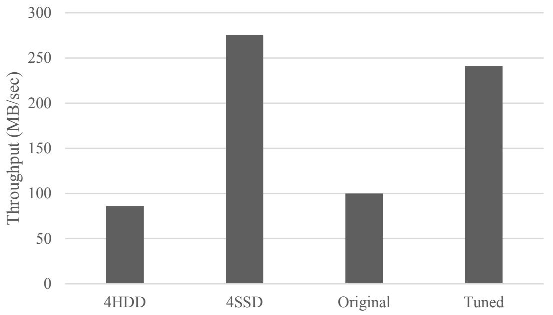

Fig. 7: Performance Comparison between Real System and Turned Simulation System

S-sim-ran are simulated results for HDD- and SSD-servers. Each client represents one IOR process, which intensively issues random write requests to the servers. The lowest error rate in these experiments is 2%. The absolute average error rate is about 11.98%.

C. Simulation Performance Study

HPIS3 can scale up by increasing the number of processes. Each node on HEC cluster has eight cores, and this study runs eight processes on each node. In total, the experiments involve 32 nodes. Figure 6 illustrates the results of the performance study of the simulator. In all the series of our experiments, all clients issue the file open, read and close requests concurrently and each node has one disk for storage. In addition, the number of clients is set to 2048, the number of servers and storage disks is set to 1024. Two major characteristics of the simulator are recorded: event rate and running time. As the number of processes scales up from 2 to 256, HPIS3 shows good scalability. In the results, the minimal event rate between these experiments is 10082.1 events/sec and the maximum is 65438.4 events/sec.

D. Hybrid PVFS Performance Tuning

To demonstrate HPIS3 as a co-design tool that is capable of tuning hybrid parallel I/O and storage system, we designed an example use case where an 8-server system needs to be optimized for better performance. In Figure 7, the "Original" bar represents the system performance before tuning on HPIS3. Here we have 4 HDD-servers and 4 SSD-servers. The "4HDD" bar shows the system performance with only 4 HDD-servers, and "4SSD" bar shows the system performance with only 4 HDD-servers. By adding 4 additional SSD servers to a 4-HDD-server system, the overall throughput has increased approximately by 15% before tuning. This performance is much worse than using only SSDs as file server, however, storage capacity is guaranteed with the HDD deployment. We modify the default striping algorithm through issuing more sub-requests to SSD nodes in order to get even response time for each sub-request. In this experiment, 16 IOR clients issue 64K random write requests to servers. As shown in Figure 7, the system throughput after tuning on HPIS3 has increased by 140%. This result is almost as high as using all SSD as file servers.

We have conducted many more experiments targeting different parameters in the hybrid systems. The experiment results show HPIS3 is effective and efficient in tuning overall system performance. Due to limited space, these experiment details are not presented in this study.

V. Conclusions and Future Work

This study presents the HPIS3 simulator: a hybrid parallel I/O and storage simulation system. HPIS3 consists of models for PVFS clients, PVFS servers, HDDs and SSDs. We validate HPIS3 through extensive real world experiments. The results show that the minimum error rate is around 2% and the average is about 11.98% in IOR tests. We demonstrate the scalability of HPIS3 by varying the number of MPI processes from 2 to 256. HPIS3 is designed to find the best configuration in a HDD, SSD, or a hybrid HDD/SSD environment, especially the latter. Our preliminary result has shown its usefulness on tuning the performance of tiered-SSD settings under PVFS.

Although HPIS3 currently focuses on HDD/SSD hybrid configurations of server nodes, there are many interesting future directions. First, more evaluation on buffered-SSD and tiered-SSD configurations of PVFS can be conducted. Second, the accuracy of HPIS3 can be improved by using more detailed models of PVFS, HDD, and SSD. Third, to improve the functionalities of HPIS3, we plan to simulate other PFSs. Finally, it will be interesting to simulate clients with hybrid configurations, power consumption and fault tolerance of storage system.

Acknowledgment

The authors would like to thank all the anonymous reviewers from PDSW'14, and our shepherds: Prof. Xiaosong Ma and Prof. Hong Jiang for providing very helpful advices to revise and improve this paper. We would also like to thank Adnan, who helped to proofread this paper. This work is supported in part by a research grant from Huawei Technologies Co, Ltd., the US National Science Foundation under Grant No. CNS-0751200, CCF-0937877 and CNS-1162540.

References

[1] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. PVFS: A parallel file system for linux clusters. In *In Proceedings of the 4th Annual Linux Showcase and Conference*, pages 391–430, 2000.

[2] Philip Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, volume 2003, 2003.

[3] Frank B. Schmuck and Roger L. Haskin. GPFS: A shared-disk file system for large computing clusters. In *FAST*, volume 2, page 19, 2002.

[4] David Nagle, Denis Serenyi, and Abbie Matthews. The panasas ActiveScale storage cluster: Delivering scalable high bandwidth storage. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, SC '04, pages 53–, Washington, DC, USA, 2004. IEEE Computer Society.

[5] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320, 2006.

[6] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. SFS: Random write considered harmful in solid state drives. In *Proc. of the 10th USENIX Conf. on File and Storage Tech*, 2012.

[7] Adrian M. Caulfield, Laura M. Grupp, and Steven Swanson. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. In *ACM Sigplan Notices*, volume 44, pages 217–228, 2009.

[8] Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, and Carlos Maltzahn. On the role of burst buffers in leadership-class storage systems. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–11, 2012.

[9] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating server storage to SSDs: analysis of tradeoffs. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 145–158, 2009.

[10] Yonggang Liu, R. Figueiredo, Yiqi Xu, and Ming Zhao. On the design and implementation of a simulator for parallel file system research. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–5, 2013.

[11] E Molina-Estolano, C Maltzahn, J Bent, and S A Brandt. Building a parallel file system simulator. *Journal of Physics: Conference Series*, 180:012050, July 2009.

[12] Taeho Kgil and Trevor Mudge. FlashCache: a NAND flash memory file cache for low power web servers. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 103–112, 2006.

[13] Timothy Pritchett and Mithuna Thottethodi. SieveStore: a highly-selective, ensemble-level disk cache for cost-performance. *ACM SIGARCH Computer Architecture News*, 38(3):163–174, 2010.

[14] Xuechen Zhang, Ke Liu, Kei Davis, and Song Jiang. iBridge: Improving unaligned parallel file access with solid-state drives. In *Proceedings of the 2013 IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS*, volume 13, 2013.

[15] Qing Yang and Jin Ren. I-cash: Intelligently coupled array of SSD and HDD. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 278–289, 2011.

[16] Eric Anderson, Michael Hobbs, Kimberly Keeton, Susan Spence, Mustafa Uysal, and Alistair C. Veitch. Hippodrome: Running circles around storage administration. In *FAST*, volume 2, pages 175–188, 2002.

[17] Guillermo A. Alvarez, Elizabeth Borowsky, Susie Go, Theodore H. Romer, Ralph Becker-Szendy, Richard Golding, Arif Merchant, Mirjana Spasojevic, Alistair Veitch, and John Wilkes. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems (TOCS)*, 19(4):483–518, 2001.

[18] Babak Behzad, Huong Vu Thanh Luu, Joseph Huchette, Surendra Byna, Ruth Aydt, Quincey Koziol, and Marc Snir. Taming parallel i/o complexity with auto-tuning. In *Proceedings of 2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2013)*, 2013.

[19] Matthieu Dorier, Gabriel Antoniu, Robert Ross, Dries Kimpe, and Shadi Ibrahim. CALCioM: Mitigating i/o interference in HPC systems through cross-application coordination. In *IPDPS-International Parallel and Distributed Processing Symposium*, 2014.

[20] Christopher D. Carothers, Kalyan S. Perumalla, and Richard M. Fujimoto. Efficient optimistic parallel simulations using reverse computation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 9(3):224–253, 1999.

[21] N. Liu, C. Carothers, J. Cope, P. Carns, and R. Ross. Model and simulation of exascale communication networks. *Journal of Simulation*, 6(4):227–236, November 2012.

[22] Ning Liu, Christopher Carothers, Jason Cope, Philip Carns, Robert Ross, Adam Crume, and Carlos Maltzahn. Modeling a leadership-scale storage system. In *Parallel Processing and Applied Mathematics*, pages 10–19. Springer, 2012.

[23] Jason Cope, Ning Liu, Sam Lang, Phil Carns, Chris Carothers, and Robert Ross. Codes: Enabling co-design of multilayer exascale storage architectures. In *Proceedings of the Workshop on Emerging Supercomputing Technologies*, 2011.

[24] Gengbin Zheng, Gunavardhan Kakulapati, and L.V. Kale. BigSim: a parallel simulator for performance prediction of extremely large parallel machines. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, April 2004.

[25] Yanlong Yin, Surendra Byna, Huaiming Song, Xian-He Sun, and Rajeev Thakur. Boosting application-specific parallel i/o optimization using IOSIG. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 196–203. IEEE, May 2012.

[26] Shuibing He, Xian-He Sun, and Bo Feng. S4d-cache: Smart selective SSD cache for parallel i/o systems. In *International Conference on Distributed Computing Systems (ICDCS)*, 2014.

[27] Shuibing He, Xian-He Sun, Bo Feng, Xin Huang, and Kun Feng. A cost-aware region-level data placement scheme for hybrid parallel i/o systems. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8, September 2013.

[28] William Loewe, T McLarty, and C Morrone. Interleved or random (IOR) benchmarks, 2007.