Accelerating a Burst Buffer via User-Level I/O Isolation

Jaehyun Han*, Donghun Koo†, Glenn K. Lockwood‡, Jaehwan Lee†, Hyeonsang Eom* and Soonwook Hwang§
*Seoul National University, jaehyunhan@snu.ac.kr, hseom@cse.snu.ac.kr
†Korea Aerospace University, amainlog@gmail.com, jlee@kau.ac.kr
‡Lawrence Berkeley National Laboratory, glock@lbl.gov
§Korea Institute of Science and Technology Information, hwang@kisti.re.kr

***Abstract*—Burst buffers tolerate I/O spikes in High-Performance Computing environments by using a non-volatile flash technology. Burst buffers are commonly located between parallel file systems and compute nodes, handling bursty I/Os in the middle. In this architecture, burst buffers are shared resources. The performance of an SSD is significantly reduced when it is used excessively because of garbage collection, and we have observed that SSDs in a burst buffer become slow when many users simultaneously use the burst buffer. To mitigate the performance problem, we propose a new user-level I/O isolation framework in a High-Performance Computing environment using a multi-streamed SSD. The multi-streamed SSD allocates the same flash block for I/Os in the same stream. We assign a different stream to each user; thus, the user can use the stream exclusively. To evaluate the performance, we have used open-source supercomputing workloads and I/O traces from real workloads in the Cori supercomputer at the National Energy Research Scientific Computing Center. Via user-level I/O isolation, we have obtained up to a 125% performance improvement in terms of I/O throughput. In addition, our approach reduces the write amplification in the SSDs, leading to improved SSD endurance. This user-level I/O isolation framework could be applied to deployed burst buffers without having to make any user interface changes.**

***Keywords*-Burst Buffer; SSD; Write Amplification; Multi-streamed SSD; Supercomputing Workload**

I. Introduction

Computing power has increased tremendously in High-Performance Computing (HPC) environments, and consequently the amount of data that HPC systems must process has increased as well. However, most supercomputers are still using Hard Disk Drives (HDDs)-based parallel file systems (PFSs) for their scratch storage, causing slow I/O to become an increasing bottleneck [1]. To alleviate this bottleneck, burst buffer technologies are being developed. These burst buffers are commonly located between compute nodes and parallel file systems (PFSs) and use fast non-volatile storage technologies such as Solid-State Drives (SSDs) to mitigate bursty I/O operations. At the time of writing, when multiple commercial burst buffers are available in some supercomputers, they are used to accelerate their I/O performance.

Flash blocks in an SSD have a block erasure characteristic; even though random page-level I/O is possible with single access, data can only be erased in the block unit. To secure empty blocks, SSD controllers periodically perform garbage collection where valid pages in blocks are copied to a new block, and then the old block is fully erased. Since garbage collection causes redundant write operations to copy valid pages, write amplification is expected in the SSD, and thus SSDs run slowly after being excessively used. SSDs in a burst buffer are particularly prone to being used excessively because the purpose of the burst buffer is to absorb intense bursts of heavy I/O operations. In addition, most of currently deployed burst buffers are share resources, and thus multiple users simultaneously perform I/O to the same SSD. Therefore, due to a large number of I/O operations in supercomputing applications, SSDs in burst buffers can suffer performance losses due to the contention [2].

We propose a user-level I/O isolation framework to minimize the garbage collection overhead. With this user-level I/O isolation framework, the burst buffer assigns each flash block exclusively to one user. Because the I/O characteristics of supercomputing workloads are different for each user's application, we expect internal fragmentation could be minimized through user-level I/O isolation. To isolate a user's I/Os from the others' I/Os, we use multi-streamed SSDs [3] which allocate the same flash block for I/Os in the same stream. The latest NVM Express specification includes the multi-stream feature as a standard [4], and thus we expect this feature will be commonly available in future high-end SSD devices. Using this multi-stream feature, we provide an exclusive stream for each user and implement the user-level I/O isolation framework straightforwardly.

To demonstrate the effectiveness of this user-level I/O isolation, we made a minimal burst buffer configuration as an experimental environment. In the experimental burst buffer, we generated supercomputing I/O patterns by using open-source supercomputing workloads and replayed I/O traces obtained from the Cori supercomputer at the National Energy Research Scientific Computing Center (NERSC). To replay the I/O traces we developed a workload emulator which generates I/O loads analogous in a different environment. With these experiments, we showed that our user-level I/O isolation framework is effective and can lead to improvements in the performance of an SSD.

Our contributions are as follows:

- Uncovering the expected performance reduction in an SSD-based burst buffer
- Presenting a method for measuring the I/O performance in an HPC environment by using open-source supercomputing workloads and I/O traces of real workloads
- Proposing a user-level I/O isolation framework for a burst buffer by using a multi-streamed SSD

The rest of the paper is organized as follows: Section 2 presents the background and our motivation of this work presented in this paper. Section 3 shows the design of our user-level I/O isolation framework. Section 4 describes the characteristics of supercomputing workloads we used. Section 5 shows the result of the performance evaluation for this work. Section 6 presents discussions on real-world burst buffers and a possible different approach, and Section 7 concludes the paper.

II. Background and Motivation

A. Burst Buffer

Traditionally HPC systems often use PFS which are comprised of only HDDs. However, with a sharp increase of data generation caused by massive new HPC systems and high-resolution experimental data sources, HDD-based PFSs will not be able to satisfy the I/O requirements of HPC in the near future [5]. Recently, the HPC community has begun deploying burst buffers to address this problem [6], [7], [8]. The main idea of these burst buffers is to augment the HDD-based PFS with a high-performance SSD tier to process burst I/Os efficiently. Various architectures of burst buffer are suggested, depending on where the burst buffer is located. Most commonly a burst buffer is located in the middle of PFS and compute node, and two state-of-the-art commercial burst buffers, Cray's DataWarp [9] and DDN's IME [10], use this architecture, and several other experimental burst buffer designs leverage this design [11]. There are other studies focusing on implementing burst buffers in a local node [12], [13], [14] and integrating burst buffers directly into a PFS.

In this paper, we focus on the architecture of Cray's DataWarp solution because it is deployed in Cori supercomputer at NERSC [7], and we used Cori's workload traces to validate our approach. As we mentioned before, in this architecture, a burst buffer is located between compute nodes and PFS as shown in Figure 1. Each burst buffer node consists of two 3.2TB SSDs, and the entire burst buffer consists of 288 burst buffer nodes. When a user requests a specific size of burst buffer space, then resource manager allocates one or multiple burst buffer nodes with a fixed ratio of capacity requested to the burst buffer nodes. As a result, a single SSD on Cori can be shared by dozens of users at the same time.

[1]Figure excerpted from a document on the NERSC burst buffer architecture [7]

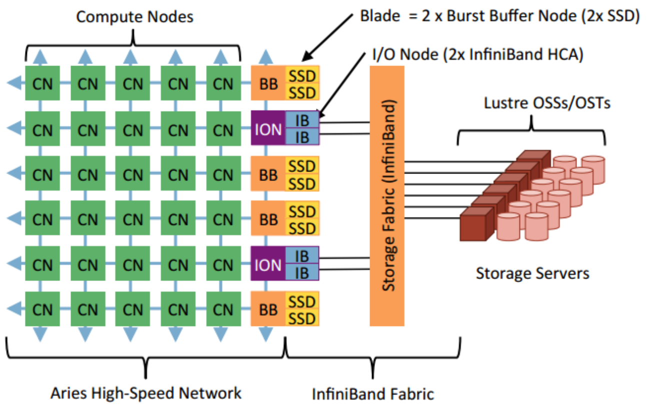


Figure 1: Burst buffer in Cori supercomputer[1]

B. Block Erasure Characteristic of Solid-State Drives

SSDs are rapidly replacing HDDs due to their fast data access time. Although SSDs can read or write data at the page level, data can only be written to pages that have been erased, and data in SSDs can only be erased at the block level. Therefore, if some pages in a block are valid, then the block cannot be erased until all of its pages become invalid. After being excessively used, a situation can arise where no erased pages are available on the whole SSD even if many pages are invalid. To write a new page, a garbage collection process inside the SSD's controller collects valid pages into the same block, then erases the invalid blocks. In this case, multiple write operations occur inside the SSD to accommodate a single host write operation, and this phenomenon is called *write amplification* [15]. When the write amplification is high, the performance of the SSD becomes low because of superfluous copy operations. The write amplification factor (WAF), calculated by Equation 1, is regularly measured to evaluate the performance of an SSD. In Equation 1, *# of NAND data units written* denotes the amount of data physically written to the NAND which includes the amount of data copied during the garbage collection.

$$\mathrm{WAF} = \frac{\text{\# of NAND data units written}}{\text{\# of Host data units written}} \tag{1}$$

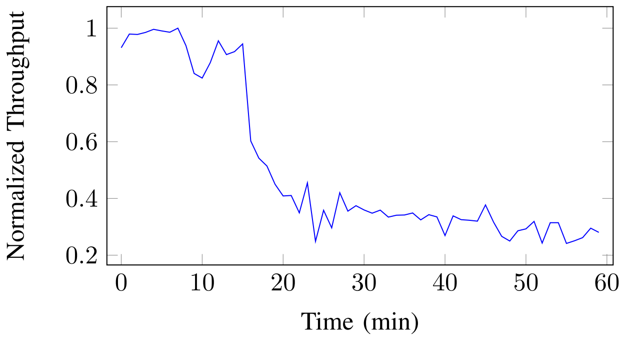
Multi-streamed SSDs have been developed to mitigate this write amplification and performance loss problem by allowing a user to allocate a stream for data so that data with similar lifetimes can be stored in the same block.

C. Motivation

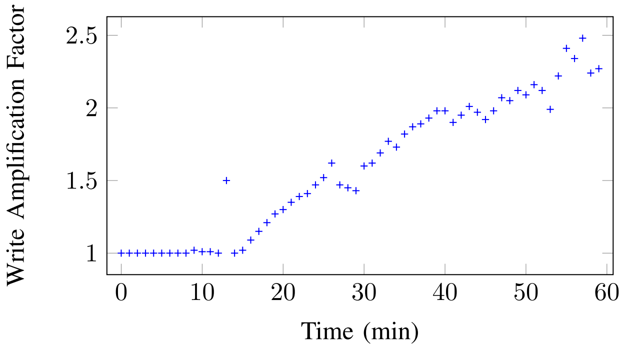
As we mentioned in Section II-A, a burst buffer is a shared resource in an HPC environment, and in the DataWarp system deployed at NERSC, dozens of users can share a single SSD. We argue that if many users simultaneously write and erase data in an SSD, then the write amplification of the SSD will increase. Because the purpose of the burst buffer is to mitigate bursty I/Os, multiple users performing bursty I/Os to each SSD is a sufficient condition to increase the write amplification of the SSD. As write amplification of the SSD becomes high, the SSD will slow down due to superfluous write operations, and this performance loss will impact the performance of the entire burst buffer. Therefore by isolating user I/Os, we can mitigate write amplification in the SSD by placing the data with a similar lifetime in the same block via preventing each block being shared by multiple users.

D. SSD impact on Burst Buffer

As a motivating example, we performed excessive I/Os in an SSD. The goal of this experiment was to show the performance degradation of SSD is a real problem in burst buffer. To focus on the SSD, we used a server node with a single SSD attached. We generated multiple supercomputing I/O workloads into the SSD to validate our argument. Before each experiment, we performed a factory reset to delete the flash mapping table and erase every block in the SSD. The results are shown in Figure 2. Since we reset the SSD before the test, the throughput of the SSD was the highest initially. However, as the time passed, the throughput suddenly dropped as the drive hit the characteristic write cliff. At the same time, as shown Figure 2(b), the WAF of the SSD increased. As we discussed above, this result is an anticipated one; as the write amplification of the SSD increases, the performance decreases.



(a) Aggregated throughput of the SSD



(b) Write Amplification Factor of the SSD

Figure 2: Aggregated throughput and Write Amplification Factors of the SSD over time

III. Design of User-level I/O Isolation System

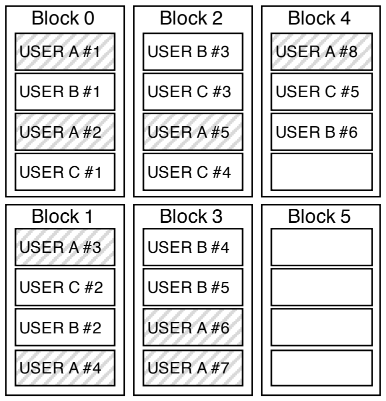
To mitigate the overhead incurred by garbage collection operations, we isolate each user's I/Os in separate SSD blocks. The lifetimes of data items in an SSD block are an important factor in data fragmentation. We propose user-level I/O isolation, utilizing the fact that the data items belonging to the same user should tend to have similar lifetimes. In this section we describe our I/O isolation method and propose a way of implementing our framework in the real-world burst buffers.

A. User-level I/O isolation using a multi-streamed SSD

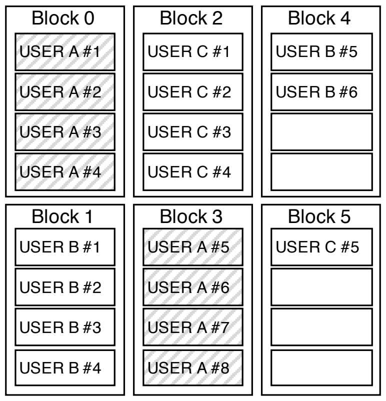
In order to isolate a user's I/Os from others', we use a multi-streamed SSD. In this multi-streamed SSD, flash blocks are exclusively used by the same streams, and the stream information is provided by the application. In a burst buffer node, the data from a different user can be clearly separated, and thus we assign each user a different stream in the burst buffer node to achieve this I/O isolation.

Figure 3 illustrates the effect of user-level I/O isolation. We assume that three users write data to an SSD simultaneously, and at a certain point user A deletes all of user A's files (for example, after those files have been visualized in transit [16]). Figure 3(a) shows an SSD in a legacy burst buffer, and Figure 3(b) shows an SSD in a burst buffer with user-level I/O isolation under this workload. In both cases, there are eight deleted pages (indicated by shaded boxes) with five pages left free. However when performing user-level I/O isolation, Block 0 and Block 3 can be erased by a single operation without any extraneous page copies made. This operation will free eight more pages. As a result of this user-level I/O isolation, 13 free pages will be available in the SSD after only two erase operations. Some pages in blocks could be wasted in a multi-streamed SSD because each stream exclusively uses a block. But block size of the SSD is smaller than 5MB [17], and half-used blocks exist only in single current block per stream. Therefore such space wastages are negligible.

To test the impact of this user-level I/O isolation, we configured a single burst buffer node with a multi-streamed SSD, and then replayed I/O traces derived from HPC workloads, to the SSD in our simulation environment. Although we used a single machine for this burst buffer, we generated multiple I/O workloads simultaneously to reproduce the multi-user workload, which a real burst buffer node would experience from the SSD's point of view. The applications we used to generate supercomputing I/O workloads are



(a) Legacy I/Os



(b) Isolated User I/Os

Figure 3: Block status of an SSD when 3 users simultaneously write data

described in Section IV.

B. *User-level I/O isolation framework*

Ultimately, we are proposing our framework for existing supercomputing environments. Figure 4 shows a current supercomputing environment. To perform a computation, a user requests resources from the system workload manager. This workload manager, which monitors the status of resources such as compute nodes and burst buffers, then reserves resources and launches the user's job within the supercomputing environment. Our framework could be seamlessly implemented in the existing environment, but we need to modify the workload manager and burst buffer to be aware of multi-streams.

The workload manager is already designed to handle streams because it is already responsible for scheduling the burst buffer requests from users. Thus, when the workload manager allocates burst buffer resources and creates the user's virtual file system, it would simply have to map a stream ID corresponding to the user to that user's burst buffer resource allocation. In this mapping, any I/O that occurs on that user's burst buffer allocation can also contain the user's stream ID, and this information would be passed down to the SSD devices from or to which the data was being read or written, respectively. Because this framework performs I/O isolation entirely within the burst buffer nodes, no changes to the user application would have to be made, and all applications will be able to realize the benefits of this optimization.

IV. Supercomputing Workloads

To evaluate our system, we used two open-source supercomputing workloads and I/O traces from the Cori supercomputer at NERSC. In this section, we introduce workloads and a workload emulator we used to evaluate the performance.

A. *Chombo*

Chombo [18] is a high-performance block-structured adaptive mesh refinement framework for solving partial differential equations in complex geometries. The EBAMRINS [19] application used in this study is built on Chombo and solves the incompressible Navier-Stokes equations on non-rectangular domains. Chombo itself scales from a single node to hundreds of thousands of cores, and its output files can range from several GB to several TB per time step. Chombo uses the HDF5 library to write and read both its checkpoint and plot data to a single shared file in parallel through MPI-IO and can make use of the burst buffer for both checkpointing and in-transit visualization [16].

B. *Nyx*

Nyx [20] is an N-body hydrodynamic cosmological simulation application that uses the BoxLib [21] adaptive mesh refinement framework. It generates separate checkpoint and plot files at each time step and, unlike Chombo, writes to multiple separate files in parallel with a configurable ratio of compute processes to I/O processes. The process of generating these files is very I/O intensive, and Nyx was one of the first extreme-scale scientific applications to be optimized to make use of burst buffers [7].

In this study, we simulate the formation of the Santa Barbara cluster [22], a standard code comparison test for cluster formation, using both the full dataset (SantaBarbara) and a lower-resolution dataset (MiniSB). To more I/O bursts while keeping wall times manageable, we configured these tests to generate checkpoint and plot files after every step. The number of files and overall size of them are listed in Table I.

C. *Cori I/O traces*

Darshan [23] is a lightweight I/O profiling tool that transparently records bounded statistics about applications' I/O behavior by intercepting I/O calls at many layers including POSIX, MPI-IO, and HDF5. Because it defers data reduction, compression, and storage until the end of

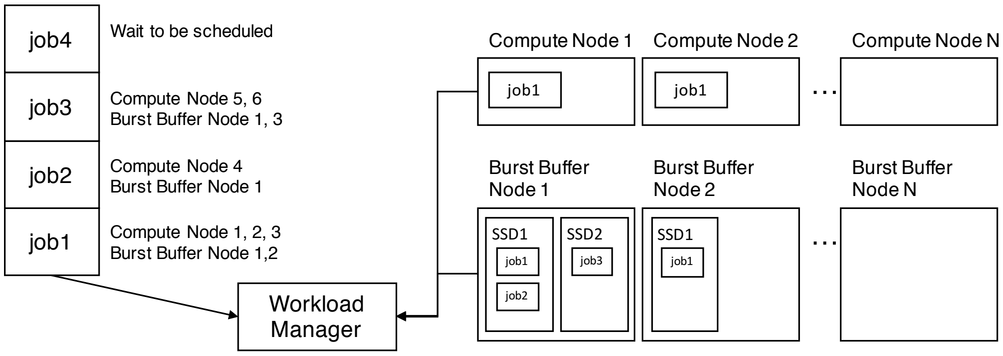


Figure 4: Architecture of a supercomputing environment

Table I: Characteristics of open-source supercomputing workloads

Workload	Number of checkpoint files	Checkpoint size (MB)	Number of plot files	Plot size (MB)
EBAMRINS - Small	1	41	1	89
EBAMRINS - Large	1	164	1	353
MiniSB	23	200	13	294
SantaBarbara	11	1400	9	1200

an application run, it is extremely lightweight and can be deployed across an entire system with minimal impacts on application performance. As a result, a significant body of application I/O data can be captured from production HPC workloads, and this study used a sampling of such profiling data from the Cori system at NERSC.

D. *Workload Emulator*

We developed a workload emulator which replays I/Os from I/O traces. Each I/O record in an I/O trace from Darshan consists of the I/O size, a timestamp at which the I/O began, and the number of I/Os. With this information, the workload emulator generates realistic application I/O patterns. Even though Darshan records I/Os from multiple layers, we focused on POSIX I/Os because most modern file systems (including DWFS) implement other I/O layers on top of POSIX, making the POSIX Darshan data the most accurate representation of the load that the SSDs experience. Because Darshan does not record the file unlinking information, we manually removed generated files at certain times while performing experiments to represent old checkpoint files being discarded. I/Os from all over the nodes were recorded in the I/O trace, and the workload emulator creates a thread for each node to emulate I/Os. In addition, the emulator can choose the maximum number of nodes to emulate to scale the workload to the size of our test system. The design of this workload emulator is illustrated in Figure 5.

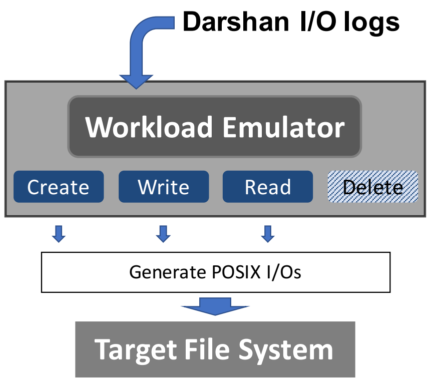


Figure 5: Workload Emulator Architecture

V. Evaluation

A. *Experimental Environments*

To validate our I/O isolation framework, we used a single node with 2-way 10-core Intel Xeon processors and 80GB RAM. For a target storage, we used a 480GB Samsung PM953 SSD, directly attached via a PCIe Gen3 bus, with modified firmware to support the multi-stream feature.

Before every experiment, we securely erased the SSD to prevent the WAF values measured in previous experiments from affecting the current measurement. While each test was running, we used `nmon` [24] to record the I/O throughput of the SSD. We also periodically calculated the WAF of the device by measuring the amount of data written by the host and the amount of data physically written to the NAND. After sufficient time, the I/O throughput and WAF reached a steady state. Once this happened, we used the averages of the values of steady-state I/O throughput and the WAF to collect the results presented.

B. *Performance under various configurations*

In order to understand the performance characteristics of our framework, we also compared the performance of the legacy system to our user-level I/O isolation system in various configurations. In these experiments, we generated bursty I/Os on an SSD using Flexible IO Tester (FIO). We then measured the throughput of the SSD while changing the space usage of the SSD, space allotment method to users, and the number of users. In these configurations, the total number of threads were the same in the experiments; as the number of users was increased, the number of I/O threads per user was reduced.

We performed two sets of experiments to evaluate the throughput of the SSD:

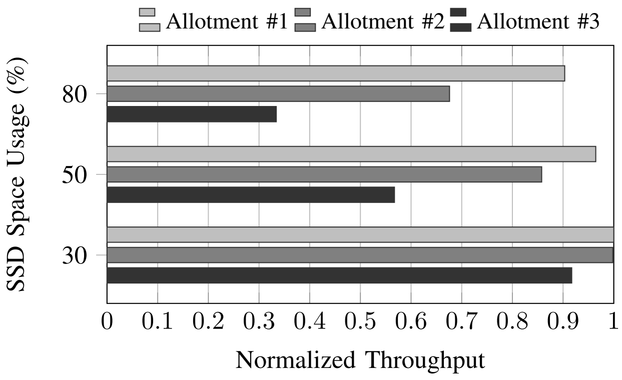
1) Performance evaluation while changing the space usage and space allotment method
   - Number of FIO threads: 24
   - Number of users: 8
   - SSD space usage: 30%, 50%, 80%
   - SSD allotment methods: Allotment #1, Allotment #2, Allotment #3
2) Performance evaluation while changing the number of users
   - Number of threads: 24
   - Number of users: 2, 4, 8
   - SSD space usage: 80%
   - SSD allotment method : Allotment #2

In both sets of experiments, FIO created 24 threads, with each generating a 64MB file continuously. We controlled the space usage of the SSD by the users' deleting their old files when their space quotas exceed the certain proportions. In experiment set #1, we also tested three SSD space allotment methods:

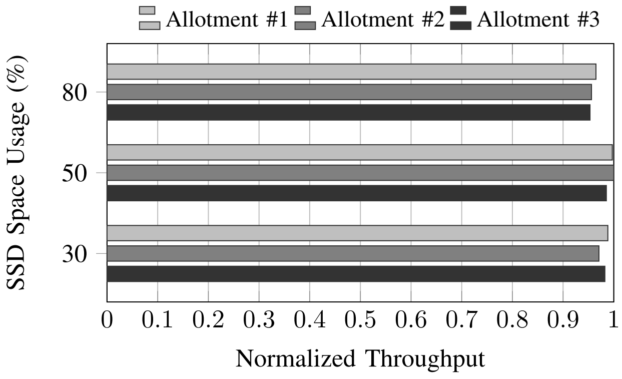
- Allotment #1 is the method that allots the same space to each user.
- In Allotment #2, users get different spaces progressively increasing from 40GB to 60GB.
- In Allotment #3, we allot bimodal spaces to users; users get spaces from 10GB to 15GB and from 60GB to 80GB.

With these allotment methods, we aim to test the effectiveness of different workloads; since all the FIO threads generate the same I/Os, a different allotment allows us to desynchronize users' I/O timing (Allotment #2) or change their I/O behavior such as how frequently they remove files (Allotment #3).

Figure 6 illustrates the normalized throughput of the SSD on both systems while changing the space usage of the SSD and space allotment method to users. On the legacy system, the throughput was decreased as more SSD space was used. This is a well-known phenomenon; due to small space left in the SSD, it becomes difficult for garbage collection operations to collect valid pages to an empty block. Moreover, the SSD controllers do not even invoke garbage collection when many free blocks are available. The throughput when using Allotment #3 shows the worst



(a) Legacy

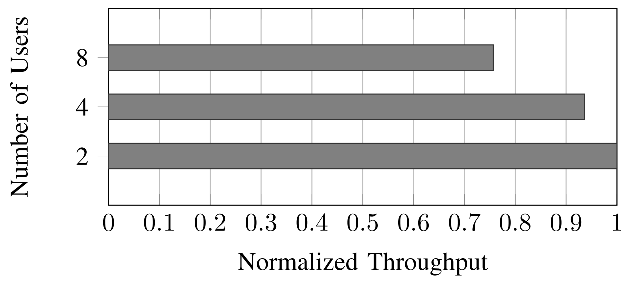


(b) User-Isolated

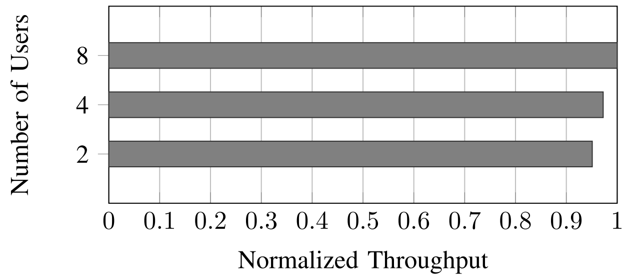
Figure 6: Normalized throughput while changing the space usage and space

performance. In this allotment method, some users have a small quota, and they undergo frequent delete operations. These frequent deletes cause many blocks to have invalid pages, and these blocks incur significant overheads since the garbage collection process must process a large number of sparse blocks. As a result, the case of 80% SSD usage with Allotment #3 shows a 67% decrease in SSD throughput. However, in the system with user-level I/O isolation, there is only a slight change in throughput because the frequent file deletions are not impacting blocks being used in other users' streams. In the worst case, only 5% performance loss is observed with user-level I/O isolation, demonstrating that our system is effective.

Figure 7 shows the normalized throughput while changing the number of users. The throughput of the SSD decreases as the number of users increases. Since all users perform I/O simultaneously, the increase in the number of users also increases the number of blocks that are shared by multiple users. In contrast, in the system with the user-level I/O isolation, the performance difference is negligible but increases slightly as the number of users increases. This slight performance decrease for small user counts results

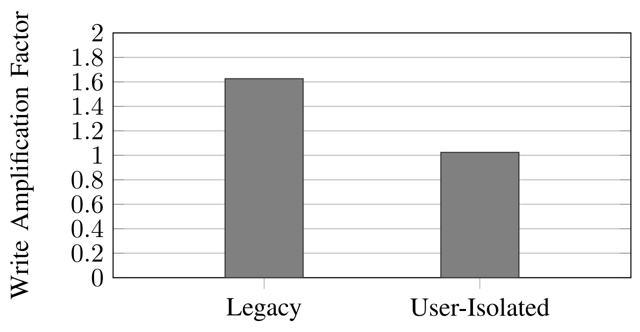


(a) Legacy

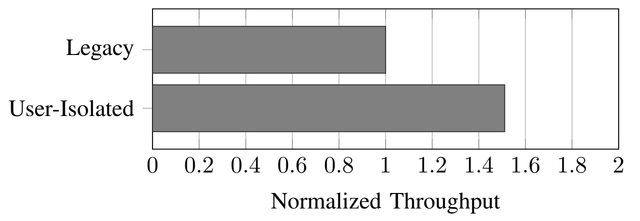


(b) User-Isolated

Figure 7: Normalized throughput while changing the number of users



(a) Average write amplification factor



(b) Average Normalized Throughput

Figure 8: Average write amplification factor and throughput of the SSD while performing experiments based on Chombo

from the overheads of our file deletion process; at small user counts, there are more files to scan and unlink when they reach their allotted quotas, which impacts our throughput measurements. Consequently, the performance decreases in order to manage generated files. In these experiments, the system with user-level I/O isolation which we are proposing retains the performance in various configurations.

In summary, the user-level I/O isolation shows better performance by reducing the garbage collection overhead, as a major factor in performance degradation, when multiple users perform bursty I/Os simultaneously.

C. Chombo

At first we performed experiments using EBAMRINS, a Chombo-based application. Table I shows the number of output files and their size for workloads we used. We configured workloads to generate one checkpoint and one plot file after each step of the computation, and we ran small version and large version of workloads which determine the size of output files.

We assume eight users are running workloads on an SSD simultaneously; Three users are running two EBAMRINS-small, and one EBAMRINS-large and the other five users are running three EBAMRINS-small workloads. Therefore each user ran three workloads. We allotted users' SSD quotas to vary in size from 30GB to 86GB. Each user performs I/Os inside their alloted quota, and when more than 90% of their space is used, the user deletes his or her files, starting with the oldest.

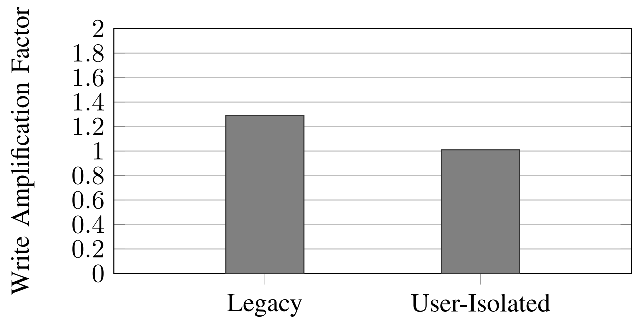
Figure 8 illustrates the resulting SSD throughput and the write amplification factor. In Figure 8(a), average write amplification factors in the legacy system shows about 1.62. This value supports our claim that superfluous write operations had occurred in the SSD. In contrast, average write amplification factor of the user-level I/O isolated system is 1.02, which show our system works as designed. Correspondingly, the throughput of SSD in our system, shown in Figure 8(b), increased by 51% compared to the legacy system.

D. Nyx

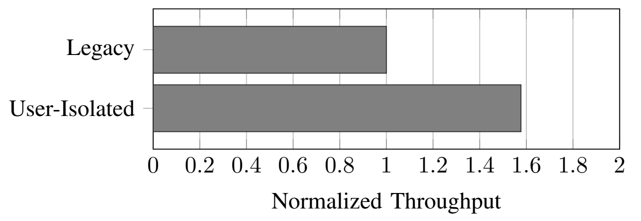
We also performed experiments using the Nyx cosmological simulation application. We ran two simulation inputs, MiniSB and SantaBarbara, which produce different output file sizes. As shown in Table I, both Nyx inputs generate tens of files for checkpointing and plotting. We configured the workload to generate a checkpoint file and a plot file at each step of the computation.

As with the Chombo experiments, we assume eight users share an SSD. We allotted users SSD spaces which vary in size from 30GB to 86GB. Each user ran one MiniSB and two SantaBarbara workloads in their own space. When a user exceeds 85% of the allotted SSD space, the user deletes files starting with the oldest first.

As shown in Figure 9, I/O throughput is improved by 58% in the user-level I/O isolated system when compared to the legacy system. In write amplification perspective, average WAF in legacy mode was 1.29 but it is improved to 1.01 by

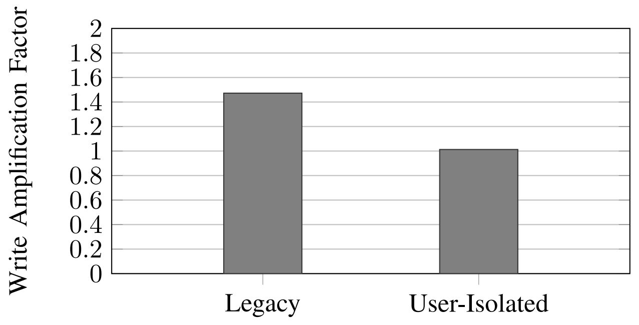


(a) Average write amplification factor

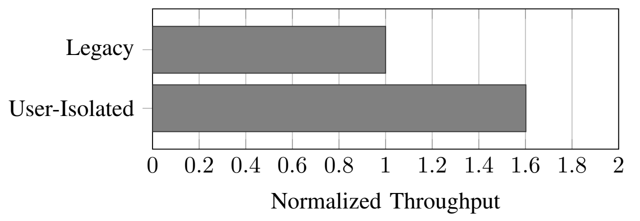


(b) Average Normalized Throughput

Figure 9: Average throughput and write amplification factor of the SSD while performing experiments based on Nyx



(a) Average write amplification factor



(b) Average Normalized Throughput

Figure 10: Average write amplification factor and throughput of the SSD while performing experiments based on mixed workloads

isolating user I/Os.

E. *Mixed workloads*

Until now, our experiments covered cases when only similar workloads were running simultaneously. From those experiments, we have demonstrated our user-level I/O isolated system is effective. However, there is a discrepancy from the real-world supercomputing environments where workloads tend to be highly diverse [25]. In this subsection, we configured the environment using multicomponent workloads composed of Chombo-EBAMRINS, Nyx, and FIO to evaluate our user-level I/O isolated system in a more realistic way.

We assume eight users are simultaneously running workloads on an SSD, with three users running Nyx, another three users running Chombo-EBAMRINS, and the remaining two users running FIO. Moreover, in this experiment, all users are differentiated by changing the number of threads or processes. We allotted the same SSD space to each user as in previous experiments, and each user performs I/Os inside his or her allotted space. Once again, users delete files in oldest order when more than 90% of their space is used.

The results are shown in Figure 10. Figure 10(a) represents average write amplification factor and shows that the average write amplification factor is 1.01 in the user-level I/O isolated system, while that of the legacy system is 1.47. This result shows even in this realistic mixed-workload configuration, our proposed system is providing consistent benefit. Average I/O throughput of the experiments is shown in Figure 10(b) and demonstrates that I/O throughput increases by 60% by effectively mitigating garbage collection overhead.

F. *Workload Emulator*

Even though our experiments above covers realistic supercomputing environments, they are synthetic configurations of workloads. As we described in Section IV-C, we obtained real workload I/O traces from the Cori supercomputer at NERSC. We selected 40 I/O traces, and for each experiment, we randomly choose 16 Darshan logs and replayed them using our workload emulator described in Section IV-D. Brief I/O characteristics of 40 I/O traces we selected are shown in Figure 11. Please note that both axes are in logarithmic scale and it shows our target workloads have widely different characteristics. We assume each user is running a workload; therefore in total 16 users shares an SSD. Although each I/O trace contains I/Os from all the nodes which the application used in the supercomputer, each workload emulator job replays I/Os of 4 nodes from the I/O trace to match the scale of our experimental environment. Since we randomly chose the workloads without knowing their characteristics, we ran this experiment five times with different sets of I/O traces to make the results more trustworthy. We allotted users SSD spaces which vary in size from 12GB to 31GB. Unfortunately, Darshan traces do not record information related to file or directory removal, so we delete files when each user's disk usage is over the certain threshold.

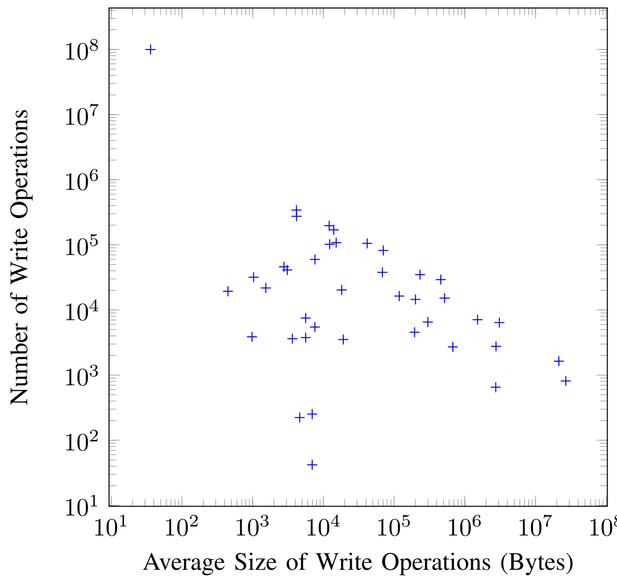


Figure 11: Characteristics of 40 I/O traces we used in workload emulator experiments

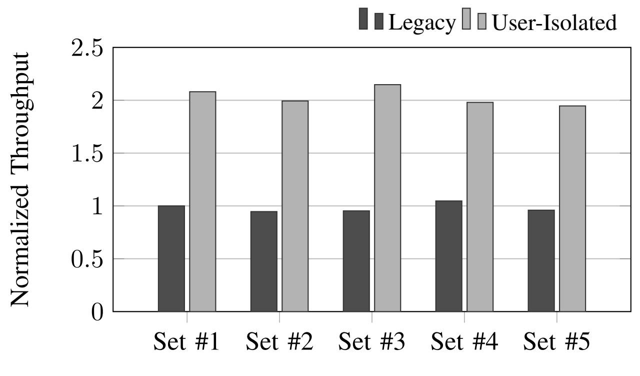
Results are shown in Figure 12. In all test cases, our user-isolated system outperforms the legacy system. Similar to previous experiments, the write amplification factor in our user-isolated system is very close to 1.0, and overall throughput improves up to 125%.

G. Summary

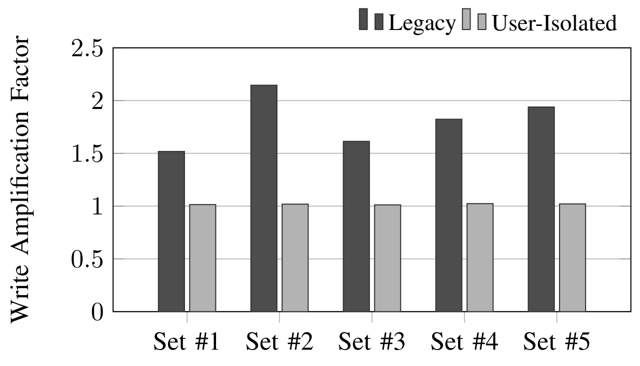
These experiments show that our user-level I/O isolation framework is useful to mitigate garbage collection overhead in SSDs. In this subsection, we summarize findings we learned from these experiments.

1) Performance over space usage and the number of users: As shown in Section V-B, the performance of SSDs decreases more when the disk usage is high. The reason is when there's plenty of space left in the SSD, we can write data to an empty block without requiring garbage collection. Also, while the same number of write operations are occurring, the number of users who delete some of their files at any arbitrary time affects the performance. This is a result of partially invalid blocks coming from different data lifetimes. We resolve this problem by applying user-level I/O isolation which works well regardless of SSD space usage and the number of users.

2) Performance when performing open-source supercomputing workloads: We performed experiments using open-source supercomputing workloads in the scenario of 8 users using a burst buffer. We used Nyx, a Chombo-based application, and FIO for generating synthetic I/Os. In these experiments, resulting average write amplification factors of the legacy system was between 1.29 and 1.62 while that of our user-level I/O isolated system was at most 1.02. This result shows our system effectively eliminates superfluous physical



(a) Average throughputs



(b) Average write amplification factor

Figure 12: Average throughputs and write amplification factors of the SSD while performing experiments using workload emulator

write operation in SSD. Correspondingly, the throughput of the SSD increased up to 60% over the legacy system.

3) Performance when emulating real supercomputing workloads: We performed experiments using a workload emulator which generates I/O based on the I/O traces collected from the Cori supercomputer. We used 40 I/O traces from real supercomputing applications and perform the evaluation five times, each time randomly choosing 16 workloads. We assume each user runs a workload, and thus 16 users share a burst buffer. The average write amplification factor of the SSD in the legacy system varied between 1.52 and 2.14. However, the average write amplification factor of the SSD in our user-level I/O isolated system was under 1.02. Again our approach successfully mitigates the overhead from garbage collection while processing bursty I/Os. As a result, the throughput of the SSD increased up to 125% over the legacy system.

VI. Discussion

A. SSDs in Deployed Burst Buffers

We analyzed internal logs of SSDs in Cori's burst buffer. Since these are parts of the running production system, we can't record device logs periodically as in our experiments,

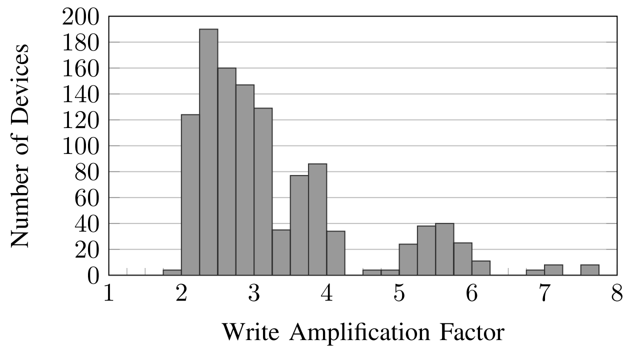


Figure 13: Write Amplification Factor of the SSDs in Cori's Burst Buffer

but with internal device logs we could calculate the average write amplification factor over the burst buffer's lifetime. We calculated write amplification factors for all devices, and as shown in Figure 13, the write amplification factors vary primarily between 2 and 4 in the SSDs in Cori's burst buffer. More than hundred devices' write amplification factors are even larger than 5, although most of these high-WAF devices were also in service for significantly less time than the majority. Even though these SSDs are different from SSDs we used in experiments, the resulting write amplification factors of our experiments in the Legacy system are between 1.5 and 2.5, and thus the real-world burst buffer suffers from significantly high write amplification as found in our evaluations. All of our results obtained with user-level I/O isolation show no more write amplification than 1.02. Therefore, we believe we can significantly reduce write amplification of SSDs in the current burst buffer without changing the user level interfaces by applying our framework presented in Section III-B. Given values of the write amplification factor, 2 to 4, we expect our framework could improve the throughput of the burst buffer more than twice, as in our experiments.

B. Flash Block Assignment by I/O Characteristics

In this paper, we achieved the performance improvement by assigning a flash block exclusively to each user. With the multi-streamed SSD, another approach is possible: users can annotate their code when performing I/Os. For example, let us assume the user is running an application which generates *checkpoint* and *computation results* as files. The user keeps the last three *checkpoint files* and deletes the older *checkpoint files*, but the user keeps all the *result files* until the application finishes. In this situation, the user can specify to use a different stream for *checkpoint files* and *result files*. In this way, data with similar lifetime could be assigned to same flash block and it is possible to separate I/O more precisely than our user-level isolation approach. In this approach the user should correctly annotate the code, which would be a burden to the user, but there is also an opportunity for checkpointing libraries such as FTL to abstract this from the application [14], [13]. Such application-level annotations would have to be supported by the burst buffer user interface; by comparison, our user-level I/O isolation is more straightforward because it does not require any user code modification and can be implemented entirely at the system level.

VII. Conclusion

With fast random access characteristic of SSDs, burst buffers now ameliorate the I/O bottlenecks of HDD-based PFSs in High-Performance Computing environments. In this paper, we argue that bursty I/Os in a burst buffer will cause write amplification in SSDs, and this causes the throughput of entire burst buffer system to degrade. With multiple supercomputing workloads, we showed that SSDs in the burst buffer are vulnerable to these bursty I/Os. To mitigate the write amplification in burst buffers, we propose a new user-level I/O isolation framework to reduce the impacts of the SSDs being a shared multi-user resource. To implement our user-level I/O isolation, we assign exclusive NAND block to each user by use of the multi-streamed SSD. Our experiments show our framework improves the throughput of the burst buffer up to 60% with open-source scientific workloads and up to 125% when replaying real I/O traces from the Cori supercomputer at NERSC. In future work, we plan to apply our user-level I/O isolation framework into actual burst buffers. In order to do this, we need to customize many parts of a supercomputing environment as described in Section III-B. Still, we expect our framework will improve the performance of the burst buffer, and thus the burst buffer will be more effective in mitigating the bottlenecks of disk-based parallel I/O.

Acknowledgment

We thank Dr. Jaeheon Jeong, Dr. Sangyeun Cho and their group at Samsung Electronics for providing the multi-streamed SSDs and helping us while using them. We also thank Katie Antypas at the National Energy Research Scientific Computing Center (NERSC) for helping us with some real applications and their burst buffer use cases on the Cori supercomputer.

This research was supported by Institute for Information and Communications Technology Promotion (IITP) grant funded by the Korean government (MSIP) (No.R0190-16-2012, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development). This work was also supported by BK21 Plus for Pioneers in Innovative Computing(Dept. of Computer Science and Engineering, SNU) funded by National Research Foundation of Korea(NRF) (21A20151113068)

References

[1] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing output bottlenecks in a supercomputer," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 8.

[2] S. Thapaliya, P. Bangalore, J. Lofstead, K. Mohror, and A. Moody, "Managing I/O Interference in a Shared Burst Buffer System," in *2016 45th International Conference on Parallel Processing (ICPP)*. IEEE, aug, pp. 416–425.

[3] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed solid-state drive." in *HotStorage*, 2014.

[4] A. Huffman, "Nvm express revision 1.3 specification," *NVM Express, Inc.*, 2012.

[5] C. S. Daley, L. Ramakrishnan, S. Dosanjh, and N. J. Wright, in *Proceedings of the 6th International Workshop on Big Data Analytics: Challenges, and Opportunities (BDAC-15).*

[6] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*. IEEE, 2012, pp. 1–11.

[7] W. Bhimji, D. Bard, M. Romanus, D. Paul, A. Ovsyannikov, B. Friesen, M. Bryson, J. Correa, G. K. Lockwood, V. Tsulaia *et al.*, "Accelerating science with the nersc burst buffer early user program," *Proceedings of Cray Users Group.*

[8] B. Hadri, S. Kortas, S. Feki, R. Khurram, and G. Newby, "Overview of the KAUST's Cray X40 System–Shaheen II," in *Proceedings of the 2015 Cray User Group*, 2015.

[9] D. Henseler, B. Landsteiner, D. Petesch, C. Wright, and N. J. Wright, "Architecture and Design of Cray DataWarp," in *Proceedings of the 2016 Cray User Group.*

[10] W. Schenck, S. El Sayed, M. Foszczynski, W. Homberg, and D. Pleiter, "Early evaluation of the "infinite memory engine" burst buffer solution," in *International Conference on High Performance Computing*. Springer, 2016, pp. 604–615.

[11] T. Wang, S. Oral, Y. Wang, B. Settlemyer, S. Atchley, and W. Yu, "BurstMem: A high-performance burst buffer system for scientific applications," in *2014 IEEE International Conference on Big Data (Big Data)*. IEEE, oct, pp. 71–79.

[12] D. Kimpe, K. Mohror, A. Moody, B. Van Essen, M. Gokhale, R. Ross, and B. R. de Supinski, "Integrated in-system storage architecture for high performance computing," in *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*. ACM, 2012, p. 4.

[13] "Design and modeling of a non-blocking checkpointing system," in *2012 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, nov, pp. 1–10.

[14] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka, "FTI: high performance Fault Tolerance Interface for hybrid systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*. New York, New York, USA: ACM Press, p. 1.

[15] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM, 2009, p. 10.

[16] A. Ovsyannikov, M. Romanus, B. V. Straalen, G. H. Weber, and D. Trebotich, "Scientific Workflows at DataWarp-Speed: Accelerated Data-Intensive Science using NERSC's Burst Buffer," in *PDSW-DISCS '16 Proceedings of the 1st Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, 2016, pp. 1–6.

[17] Samsung Electronics, "Samsung V-NAND technology (White Paper)," Tech. Rep., 2014.

[18] P. Colella, D. Graves, T. Ligocki, D. Martin, D. Modiano, D. Serafini, and B. Van Straalen, "Chombo software package for amr applications-design document," 2000.

[19] G. H. Miller and D. Trebotich, "An embedded boundary method for the navier-stokes equations on a time-dependent domain," *Communications in Applied Mathematics and Computational Science*, vol. 7, pp. 1–31, 2012.

[20] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukić, and E. Van Andel, "Nyx: A massively parallel amr code for computational cosmology," *The Astrophysical Journal*, vol. 765, no. 1, p. 39, 2013.

[21] J. Bell, A. Almgren, V. Beckner, M. Day, M. Lijewski, A. Nonaka, and W. Zhang, "Boxlib users guide."

[22] C. S. Frenk, S. D. M. White, P. Bode, J. R. Bond, G. L. Bryan, R. Cen, H. M. P. Couchman, A. E. Evrard, N. Gnedin, A. Jenkins, A. M. Khokhlov, A. Klypin, J. F. Navarro, M. L. Norman, J. P. Ostriker, J. M. Owen, F. R. Pearce, U. Pen, M. Steinmetz, P. A. Thomas, J. V. Villumsen, J. W. Wadsley, M. S. Warren, G. Xu, and G. Yepes, "The Santa Barbara Cluster Comparison Project: A Comparison of Cosmological Hydrodynamics Solutions," *The Astrophysical Journal*, no. 2, pp. 554–582, nov.

[23] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale i/o workloads," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–10.

[24] N. Griffiths, "nmon performance: A free tool to analyze aix and linux performance," 2003.

[25] G. P. Rodrigo Álvarez, P.-O. Östberg, E. Elmroth, K. Antypas, R. Gerber, and L. Ramakrishnan, "HPC System Lifetime Story," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '15*. New York, New York, USA: ACM Press, pp. 57–60.