Title
An empirical study of I/O separation for burst buffers in HPC systems

Authors
Koo, Donghun
Lee, Jaehwan
Liu, Jialin
et al.

Publication Date
2021-02-01

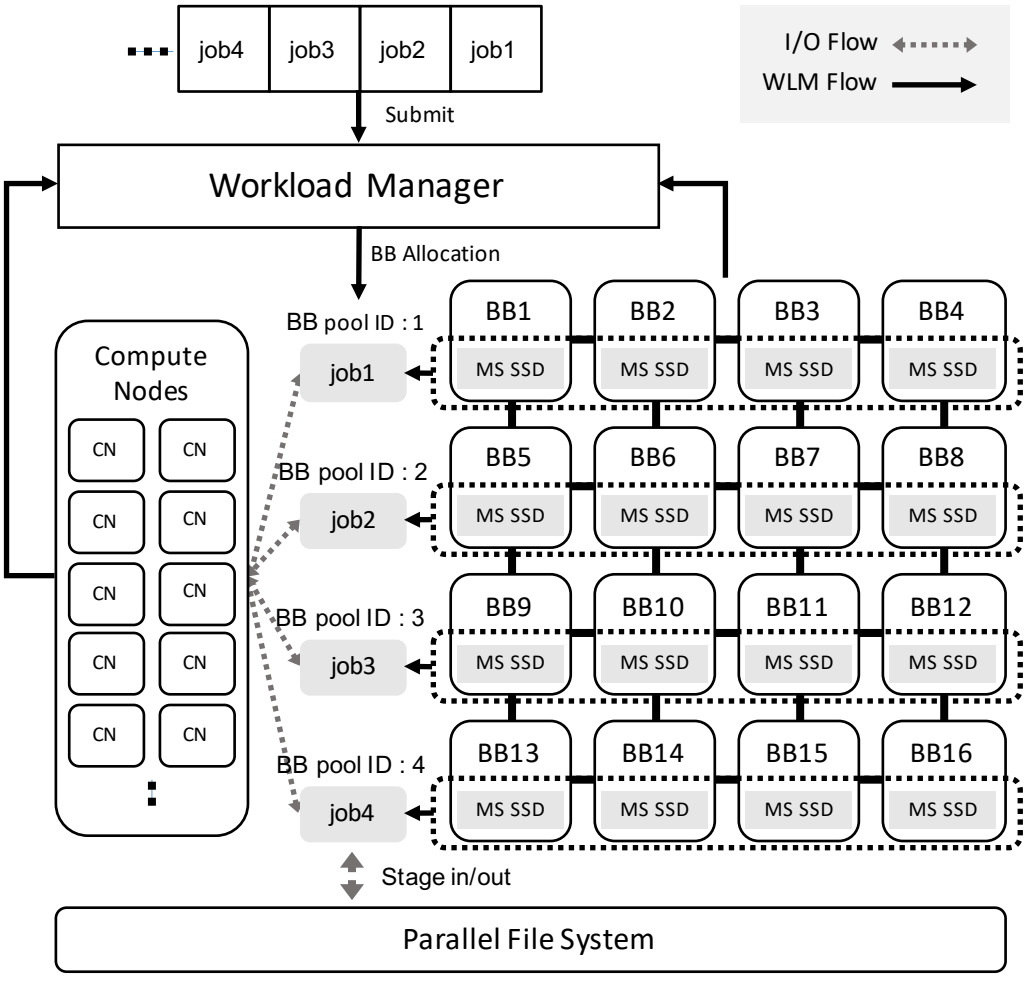
DOI
10.1016/j.jpdc.2020.10.007

Peer reviewed

Graphical Abstract

An Empirical Study of I/O Separation for Burst Buffers in HPC Systems

Donghun Koo,Jaehwan Lee,Jialin Liu,Eun-Kyu Byun,Jae-Hyuck Kwak,Glenn K. Lockwood,Soonwook Hwang,Katie Antypas,Kesheng Wu,Hyeonsang Eom



Highlights

An Empirical Study of I/O Separation for Burst Buffers in HPC Systems

Donghun Koo,Jaehwan Lee,Jialin Liu,Eun-Kyu Byun,Jae-Hyuck Kwak,Glenn K. Lockwood,Soonwook Hwang,Katie Antypas,Kesheng Wu,Hyeonsang Eom

- Develop an I/O separation scheme using multi-stream feature of solid state drives, which reduces garbage collection overheads, improves I/O throughput, and extends device lifetime.
- Design a stream-aware scheduling policy to work with the I/O separation scheme and improve overall I/O throughput.
- Improves performance without require users to change their I/O functions.
- Increases I/O throughput by 44% and reduce Write Amplification by 20%.

An Empirical Study of I/O Separation for Burst Buffers in HPC Systems

Donghun Koo[a], Jaehwan Lee[b,*], Jialin Liu[c], Eun-Kyu Byun[d], Jae-Hyuck Kwak[d], Glenn K. Lockwood[c], Soonwook Hwang[d], Katie Antypas[c], Kesheng Wu[c] and Hyeonsang Eom[a]

[a]Seoul National University, Seoul, South Korea
[b]Korea Aerospace University, Goyang, South Korea
[c]Lawrence Berkeley National Laboratory, CA, USA
[d]Korea Institute of Science and Technology Information, Daejeon, South Korea

ARTICLE INFO

Keywords:
Burst Buffer
Multi-streamed SSD
I/O Separation
Stream-Aware
Evaluation

ABSTRACT

To meet the exascale I/O requirements for the High-Performance Computing (HPC), a new I/O subsystem, Burst Buffer, based on solid state drives (SSD), has been developed. However, the diverse HPC workloads and the bursty I/O pattern cause severe data fragmentation that requires costly garbage collection (GC) and increases the number of bytes written to the SSD. To address this data fragmentation challenge, a new multi-stream feature has been developed for SSDs. In this work, we develop an I/O Separation scheme called BIOS to leverage this multi-stream feature to group the I/O streams based on the user IDs. We propose a stream-aware scheduling policy based on burst buffer pools in the workload manager, and integrate the BIOS with the workload manager to optimize the I/O separation scheme in burst buffer. We evaluate the proposed framework with a burst buffer I/O traces from Cori Supercomputer including a diverse set of applications. Experimental results show that the BIOS could improve the performance by 1.44x on average and reduce the Write Amplification Factor (WAF) by up to 1.20x. These demonstrate the potential benefits of the I/O separation scheme for solid state storage systems.

1. Introduction

Recently, the data-intensive scientific domains have become a new application field, due to the increasing volume of scientific data of up to several petabytes and the complexity of the scientific workload, the I/O has become more and more challenging. Consequently, existing HPC Parallel File Systems (PFSs) based on disk cannot satisfy the I/O requirements. In response to the demand for high I/O performance in an HPC environment, many supercomputers have been introduced with SSD-based file systems. For example, Cray has developed a DataWarp which is a fast storage tier based on NVMe SSDs, and deployed it at NERSC (National Energy Research Scientific Computing Center) since 2015[1]. As a high-performance storage layer, burst buffers[2] have effectively handled the burst I/Os in many commercial HPC systems.

Unfortunately, all burst buffers based on SSDs have the same problem arising from a characteristic of the NAND flash memory: Garbage Collection (GC) overheads caused by a difference in the operation unit between write/read (page level) and erase (block level). The GC overhead is an additional copy operation for preserving valid pages in the GC operation that secures the empty block, which adversely affects the performance and the lifetime of a flash device when frequent GC operation occur[3][4]. Specially, SSDs in burst buffer must process a large amount of concurrent and complex I/Os from a lot of scientific applications, because burst buffers are used for absorbing the bursty I/O traffic as a shared resource in HPC systems[5]. As a result, these SSD's used in such HPC environments, would be exposed to frequent GC operations, which could lead to the losses in performance and endurance [6].

In relation to research on burst buffers, the research has mainly focused on studies addressing the I/O bottleneck problems in HPC systems[2, 1, 7], on burst buffers in local nodes for scalable write performance[8, 9], on I/O scheduling in burst buffer[10, 6], or on burst buffer resource management[11]. To the best of our knowledge, no existing studies focused on improving performance and endurance of the burst buffer itself except for our own previous work[12]. To mitigate the GC overheads for SSDs in burst buffer, our previous work proposed the user-level I/O isolation by using a multi-streamed SSD[13] that allocates same flash block for I/Os in the same stream ID. In that work, we uncovered the expected performance reduction in an SSD based burst buffer and demonstrated the that effectiveness of user-level I/O isolation in the burst buffer. However, that work was not integrated into a actual burst buffer on a HPC system.

In this work, we implement the burst buffer named *BIOS*, to transparently support I/O separation, and to validate its performance in an active HPC environments. To take full advantage of this I/O separation scheme, we design a stream-aware scheduling policy for the workload manager. Based on this implemented system, we explore the benefits and limitations of the BIOS and the framework through extensive experimentation. To implement the burst buffer with I/O separation scheme, we leverage the multi-streamed SSDs to group the I/O streams based on the user stream mapping. This burst buffer is to assign the flash memory blocks exclusively to each user being performed transparently to users. In HPC burst buffer environments, we found that the benefits of I/O separation scheme can be reduced due to user ID-based stream allocation and a limited number of available streams

*Corresponding author
ORCID(s): 0000-0001-6248-9567 (J. Lee)

(e.g., 4 to 16) in a multi-streamed SSD[13],[14],[15]. To overcome these problems, we propose managing the burst buffer resource as burst buffer pools and the stream-aware scheduling policy in workload manager and finally, implement the framework by integrating these with the BIOS. This framework optimizes the benefits of I/O separation scheme by alleviating the interference caused by data striping and the problem of skewed stream allocation.

To validate the effectiveness of the BIOS framework in the HPC environment, we use not only synthetic workloads but also real HPC applications as well as burst buffer I/O traces obtained from the Cori supercomputer at NERSC. In our tests, the BIOS shows the up to 44% increase in I/O throughput and up to 20% decrease in write amplification factor (WAF) compared to an existing burst buffers.

Our main contributions are as follows: (1) We implement the burst buffer framework with an I/O separation scheme for multiple devices and multiple nodes; (2) We design an resource efficient workload manager with a stream-aware scheduling policy to take advantage of the I/O separation framework; (3) Users do not have to change their I/O functions in order to benefit from our burst buffer framework. (4) Through extensive evaluations with HPC applications and burst buffer I/O traces from the Cori supercomputer, we observe up to 44% increase in I/O throughput and 20% descrease in write amplification.

In the rest of this paper, Section 2 presents the background and challenges of the work presented in this paper and Section 3 and 4 describe the I/O separation scheme and framework. Section 5 shows the experiment results to demonstrate the effectiveness of proposed scheme and framework compared with legacy burst buffer. Section 6 summarizes findings and insights from the evaluation. Section 7 and 8 show the discussion and related work respectively, and finally Section 9 summarizes our conclusions and future studies.

2. Background And Challenges

2.1. Burst Buffer

With the emergence of massive HPC systems and corresponding dramatic increase in computing resource performance, traditional HPC systems with disk based PFSs are unable to satisfy the I/O requirements. Burst buffer technology has emerged in recent years as a result of demands for better I/O performance in HPC environments. The burst buffer as a high-performance SSD tier efficiently handles bursty I/O that is not handled by the PFSs, located between the compute node and the PFS. In accordance with the purpose of the burst buffer, it efficiently handles the traditional HPC checkpoint restart workloads which have large streaming I/Os, and recently, is also being used for temporary staging space and in-transit data processing recently[16]. However, due to these various demands, a burst buffer faces a performance challenge with small files and varied I/O patterns [1]. In HPC burst buffer, currently, there are two representative burst buffer architectures: the local burst buffer and the shared burst buffer. One is to be implemented in each compute node as a local burst buffer and the other used a shared resource

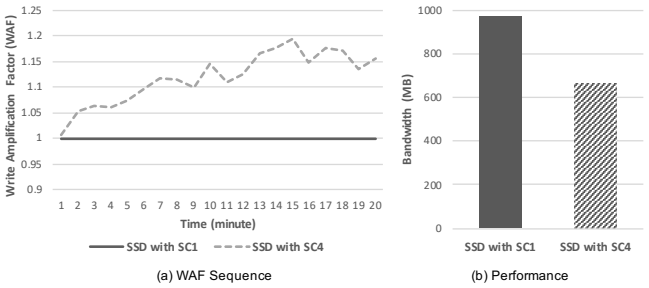


Figure 1: Impact on multi-streamed SSD when the stripe count number is different for the same load

such as dedicated burst buffer nodes. With benefits from ease of maintenance and deployment, shared burst buffer architecture is commonly used in state of the art commercial burst buffers such as Cray's DataWarp[17] and DDN's IME[18]. Specifically, on Cori supercomputer at NERSC, DataWarp burst buffer is treated as high-speed storage resources and managed by a batch scheduler, SLURM[19]. Through the workload manager, the shared burst buffer is allocated to users on a per-job or short term basis.

2.2. Write Amplification in SSDs

The NAND flash memory used as a storage for burst buffer has an inherent characteristic that erase has coarser granularity (block-level) compared to granularity (page-level) of write and read operations. To secure the free blocks, the garbage collection is performed in SSD, causing a redundant copy operation for keeping valid pages. The characteristics of out place update and differences in block erase and page write/read operations in flash memory are the cause for unnecessary copy operations. So the amount of data written to the storage media is amplified in SSD when GC operation occurs. This phenomenon is called *write amplification*[20] and the amplification ratio is denoted as write amplification factor (WAF). This write amplification factor is calculated as follows:

$$\text{WAF} = \frac{\text{the amount of data written in NAND flash}}{\text{the amount of data sent from the client}} \quad (1)$$

The cost of the erase operation is considerably expensive due to the write amplification caused by the GC operation. When an SSD is being used excessively, an SSD incurs more frequent GC operations at the same time, degrading the write performance of an SSD. Therefore, SSDs that are used as a shared resource to handle the burst I/O are more affected by the number of GC operations performed.

2.3. Multi-streamed SSD

A multi-streamed SSD has been proposed to reduce the garbage collection(GC) overheads in flash memory by mapping the data with different lifetimes to different streams. To map the data to stream ID, a multi-streamed SSD makes the host system to send write request with stream ID to an SSD since the host can provide adequate information about data lifetime. Therefore, the stream ID may be a hint about

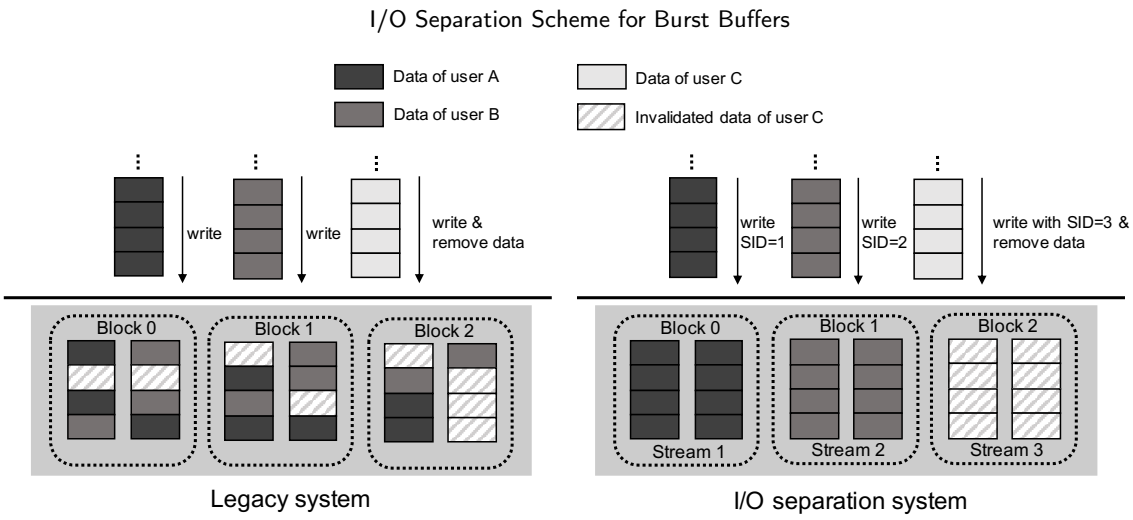


Figure 2: Legacy System vs I/O Separation System

data lifetime, which allowing an SSD to place the data with the same stream ID into the same flash blocks. With this mechanism, all data associated with stream is expected to be invalidated at the same time in the same flash blocks, and thus this feature increases the probability of the data within the blocks to be removed together, which reduces the probability of a number of copy operations during the GC operation. A multi-streamed SSD, as a result, provides not only an enhanced device lifetime but also improved performance and constant latency via the multi-stream mechanism. In the NVMe 1.3 specification[21], the multi-stream feature is introduced as a form of directives. This feature is defined for write commands, allowing the host system to carry the stream to the controller by using NVMe commands. As the multi-stream feature is officially adopted in the NVMe interface, the multi-stream feature is expected to be available in many future NVMe devices.

2.4. Challenges of Multi-stream Feature in Burst Buffers

In this search, we seek to implement a robust burst buffer from a large amount of I/Os using a multi-stream feature. In other words, by separating the user's I/Os from the others' I/Os via multi-streamed SSDs, we target to minimize the GC overheads in SSDs of burst buffers. The number of streams supported by the device, therefore, is important in reducing the number of GC overheads. For example, if multi-streamed SSD can provide an unlimited number of streams, all I/O streams can be allocated to exclusive flash blocks, which can remove the GC overheads completely. Unfortunately, a multi-streamed only SSD supports a number between 4 to 16 streams due to implementation constraints related to the write buffering mechanism in an SSD[22]. In HPC environments, as a result, each stream ID is shared by multiple I/O streams, resulting in data mixed in flash blocks associated with same stream IDs and eventually weakens the benefits of the multi-stream feature. To address this problem, most research on multi-streamed SSD has tried to devise a method for stream allocation which maps data with a similar lifetime to the stream IDs[14],[23],[15], [24]. In this paper, we present intuitive and effective criteria for stream allocation considering the burst buffer environments.

In this paper, we consider a shared burst buffer system located in a dedicated node that uses striping I/O for high performance. Assuming we build a burst buffer using multi-streamed SSDs in this system, in theory, we can use a number of streams equal to # of devices * # of supported streams to completely isolate the I/O stream from other I/O streams in this burst buffer. However, the number of streams that can actually be used would be decreased since the striping I/O segments the file and stores each segment on different SSDs. Each file uses a stream assigned to that file in all multi-streamed SSDs, which brings the effect like using only the number of streams supported by a single multi-streamed SSD when all SSDs participate in striping I/O.

To demonstrate the impact of data striping on multi-streamed SSD, we perform the workload with 8 FIO[25] threads in each of the 4 nodes to local SSD and grouped SSDs with RAID 0 respectively, and these results about the local SSD (SSD with stripe count 1) and one of the grouped SSDs (SSD with stripe count 4) after preconditioning to warming up the devices are showed in figure 1. Since we use 8 stream IDs to isolate the I/O threads, the local SSD perfectly separates the I/O threads in each node, while each SSD grouped with RAID 0 handles the 32 I/O threads due to striping I/O; all stream IDs in SSD are shared by 4 I/O threads each. As a result, WAF in SSD with SC4 is increased as time passed, adversely affecting the performance despite using the multi-stream feature. In this context, it showed that simply applying the stream allocation strategy in burst buffer is insufficient to keep up the maximum benefits of the multi-stream feature.

3. I/O Separation Scheme in Burst Buffer

In this section, we first present the intuitive and effective criterion to distinguish the data lifetime in burst buffer environments and using this criterion, we implement the burst buffer providing a multi-stream feature to user transparently,

Algorithm 1 A User ID-based Stream Management

```
1: streamID = getStreamID_from_table(userID)
2:
3: if !streamID then
4:    if Find_idle_streamID then
5:       Allocate the streamID
6:    else
7:       Allocate the least used streamID
8:    end if
9:    Register the (userID, streamID) pair to table
10: end if
11:
12: fadvise(fd, streamID)
13: update_access_time(userID)
14: threshold++
15:
16: if threshold > setting_value then
17:    Check all registered user's stream ID access time
18:    if Not_recently_accessed then
19:       Retrieve the streamID
20:       Remove the (userID, streamID) pair in the table
21:    end if
22:    Init threshold
23: end if
```

named *BIOS*. Before addressing the challenge of multi-stream feature in the burst buffer environment, it is important the implementation of an effective burst buffer must come first in order to maximize multi-stream capability.

3.1. Stream Allocation Criteria

The mapping method between data and stream ID is the most important factor in optimizing the multi-stream mechanism. In shared burst buffer systems, burst buffer is assigned to a user on a per-job basis through the workload manager. The output data of the job are stored in the burst buffer while the job is running, but both burst buffer and output data are deleted together when the job is completed. Namely, the lifetime of the user's data stored in the burst buffers is generally equal to the job execution time[26]. The user ID can be the key to intuitively and effectively distinguish the data lifetime in the burst buffer environments. Therefore, we take the *user ID* as a classification unit for mapping the data with a different lifetime to disparate stream IDs. Based on this insight, we present user ID-based stream management in burst buffer, I/O separation scheme.

To prove the effectiveness of an I/O separation scheme, we assume a situation where a user's data is deleted while other users are still writing data to burst buffer; this situation is commonly found in shared burst buffer environments. Figure 2 illustrates the effect of an I/O separation scheme compared to the legacy system. More specifically, it shows the different flash memory block layout in a legacy and an I/O separation system when GC operation is performed on block #2 respectively. In case of the legacy system, to perform the GC operation for block #2, it needs 4 copy and 1 erase operation. On the other hand, the I/O separation system can complete the GC operation with just 1 erase operation. It's intuitive that the I/O separation system can reduce the GC overheads efficiently compared to the legacy system in burst buffer environments.

3.2. Implementation

To implement the burst buffer with I/O separation scheme, we modified the open source based distributed file system, BeeGFS [27], to allow it to allocate stream IDs by leveraging user IDs, uid in Linux, and to pass the stream IDs by using *fadvise()* which passes it down to SSDs through file inode; a stream allocation is performed per file descriptor when a file is opened.

The logic for implementing I/O separation scheme is represented by algorithm 1 that describes the user ID-based stream management in the BeeGFS storage daemon located in each node. In the algorithm 1, the stream and access time mapping table are used for stream management; these are data structure for managing the stream IDs efficiently, returning the stream ID and access time corresponding to user ID respectively. When a user writes the data, a BeeGFS checks the list to see whether the user ID is registered in the stream mapping table or not. If not, the BeeGFS attempts to find the idle stream ID first. If all stream IDs are in use, it finds the least used stream ID as an alternative. Next, the BeeGFS registers the (user ID, stream ID) pair to the stream mapping table. After the stream allocation process, the selected stream ID is applied to file descriptor through the modified *fadvise()*. And then, user's stream ID access time is updated in the access time mapping table and a threshold value is incremented; these variables are used as a criterion for whether or not the stream ID will be reclaimed and to periodically perform a retrieval function respectively. When the threshold value reaches the setting value, the retrieval function is performed. The retrieval function checks the all registered user's stream ID access time, removing all data related to corresponding to the user ID such as the (user ID, stream ID) pair in the stream mapping table when user has not used the stream ID for a certain period of time. After the retrieval function is finished, the threshold value is then initialized.

In summary, the overall process of data flow from BeeGFS to multi-streamed SSDs is as follows. When the user writes the file to a BeeGFS, it divides the file into chunks for data striping, sending the chunks to nodes which belong to BeeGFS. Then, a BeeGFS storage daemon which services the storage function in all nodes belonging to BeeGFS assigns the stream ID for chunk using algorithm 1; next the chunks with stream ID are passed down to a multi-streamed SSD in each node. Then, each SSD stores the received chunk into the flash blocks associated with each stream ID. Therefore, this system, named *BIOS*, which provides an I/O separation scheme offers a multi-stream feature to users transparently, so that all users in the BIOS can experience these benefits without modifying the application's code.

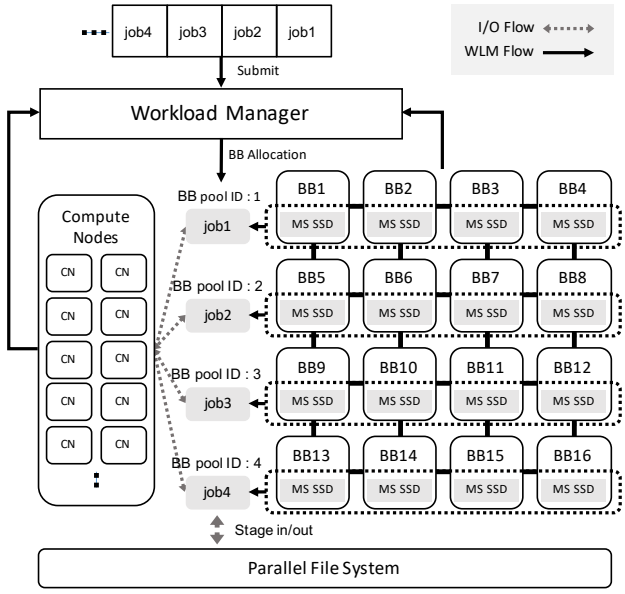


Figure 3: An overview of BIOS framework

3.3. Limitations of User ID-based Stream Allocation

As discussed above, a user ID is the most suitable information about data lifetime when we considered burst buffer environments. However, a problem can occur when the same user submits multiple jobs because the burst buffer supporting user ID-based stream allocation assigns the same stream ID to all jobs from the same user. In this case, skewed stream allocation can occur in the BIOS. In summary, the problems of skewed stream allocation along with the reduction of available streams incurred by data striping are factors which limit the benefits of I/O separation scheme.

4. BIOS Framework

In this section, we propose the burst buffer system, *BIOS Framework*, which optimizes the I/O separation scheme by integrating the BIOS with workload manager. Through integrating with the workload manager, we can treat burst buffer as high-speed storage resource, optimizing the use of burst buffer (e.g., Cray DataWarp). To improve the limitations of I/O separation scheme in the BIOS framework, we organize resource unit for burst buffer allocation by grouping the devices together in a burst buffer pools and design the stream-aware scheduling policy which evenly balances the load to SSDs while decreasing the contention of stream resource in workload manager.

4.1. Support in Workload Manager

In existing HPC systems such as supercomputers, the workload manager is an integral component for cluster/resource management and job scheduling. Most resources including the burst buffer in HPC systems, are managed and assigned to a user for a set amount of time by the workload manager. For example, in the case of ephemeral burst buffers as of NERSC Cori, the allocation process is mainly managed by the SLURM workload manager. Therefore, supporting the burst buffer in the workload manager is essential for implementation of the real burst buffer system.

Algorithm 2 A Stream-Aware Scheduling Policy

```
1:  pool_ID = get_idle_pool_id()
2:
3:  if pool_ID then
4:      Return pool_ID
5:  else
6:      user_ID = get_uid_from_job_id(jobID)
7:      pool_ID = not_overlap_pool_id(userID)
8:
9:      if !pool_ID then
10:         pool_ID = min_used_pool_id()
11:         Return pool_ID
12:     else
13:         Return pool_ID
14:     end if
15: end if
```

To implement a framework that supports the BIOS, we used the SLURM workload manager with some modification. Specifically, in order for SLURM to support the BIOS, we added the SLURM directive for the BIOS. As an argument of the directive, it includes a directory name, *dirname*, and the size of burst buffer, *capacity*. In the *dirname* argument, it's used for the directory name of burst buffer; the directory name for job is created as a "user/dirname" to prevent the problem of name duplication between the users. A *capacity* is used for the size of burst buffer and for a baseline of setting the stripe count which is the number of SSDs to use. In current burst buffer on Cori, the allocation granularity is 20G, for example, when the *capacity* is set to 80GB, the stripe count will be 4 (four burst buffer nodes will be allocated). In our BIOS framework, we configured the maximum stripe count as 4. By specifying the BIOS directive, *#BIOS*, in a SLURM batch script, users can be assigned the ephemeral burst buffer with an I/O separation scheme from the workload manager.

4.2. Burst Buffer Pools

In order to reduce the contention caused by data striping and to prevent skewed stream allocation when a lot of jobs are submitted from the same user, we utilized the storage pools function [28] in BeeGFS to split the burst buffer resources into different burst buffer pools, as shown in figure 3. By utilizing the burst buffer pools, we can limit the range of data striping and choose the physical storage device for burst buffer, which gives a chance to mitigate the contention from data striping and to solve the skewed stream allocation via scheduling technique. Figure 3 shows the BIOS framework with 4 burst buffer pool IDs composed of 16 storage devices. In this framework, the workload manager determines the pool ID first (see next subsection for details) and then creates the burst buffer within the pool for each job.

But, using a burst buffer pools may result in the inefficient use of burst buffer resources when demand for burst buffer is usually low because burst buffer resource per job is limited in burst buffer pool even if there are idle resources remaining. However, since the utilization of the burst buffer has recently been increasing [29], the inefficiency from low utilization is negligible. Next we discuss the stream-aware scheduling policy based on burst buffer pools.

4.3. Stream-Aware Scheduling Policy

Ultimately, we implement the real burst buffer system which manages the burst buffer resource by the workload manager. To efficiently manage the resource, it is important to use an appropriate scheduling scheme. For example, in burst buffer system, a round robin scheduling is used to assign it to the user[30]. In resource scheduling, the most important thing is to ensure that resources are used evenly. In order to design the scheduling policy to optimize the I/O separation scheme in the burst buffer system, we first consider distributing the load evenly to the SSDs in burst buffer pools. Moreover, we also consider the stream ID-use-state to mitigate the problem of skewed stream allocation and contention in stream ID. Based on these considerations, we propose the stream-aware scheduling policy which considers not only load balancing, but also a user ID-based stream allocation in the burst buffer system.

The algorithm 2 describes a stream-aware scheduling policy in the workload manager. When jobs are submitted, the workload manager first tries to find an idle pool and creates the burst buffer on it. If an idle pool does not exist, the workload manager allocates a pool that is not used by the same user ID by checking each stream ID-use-state. As discussed in Section 3.1, we use the user ID-based stream allocation in the BIOS, thus jobs from the same user are assigned the different pool IDs, guaranteeing that each job uses their own stream IDs on different SSDs. If all pool IDs are being used from the same user, the workload manager assigns the least used pool ID to the job in order to minimize interference as much as possible.

4.4. Workflow of BIOS Framework

Figure 3 shows the overall structure of a BIOS framework which includes the BIOS, the workload manager, and parallel file system (PFS). The BIOS framework also supports the staging function: stage-in/out which moves the data between burst buffers and PFSs. When jobs are submitted to the workload manager, it figures out the user ID of job and the all pool IDs use-state, assigning a burst buffer to the job. Before starting the job execution, stage-in is performed when the user demands this function; stage-in is necessary if files of frequently accessed or large size are needed. After stage-in, a job starts to run. When output files are written to an allocated burst buffer after job execution, BeeGFS storage daemon in each node assigns the stream ID to the output files by identifying the user ID. Then, when the job is terminated, the workload manager performs the stage-out transferring the files stored in the burst buffer to the PFS and terminates the job by removing the burst buffer. The above process is simply performed by specifying a BIOS directive in the SLURM batch script. As a result, the BIOS framework we implement provides the service of a complete burst buffer system.

5. Evaluation

5.1. Experiment Setup

For conducting the experiments, we used the testbed which consists of 4 nodes. Each of its nodes has an Intel Xeon E5-2620 v4 processor with 2-way 8-core and 64GB RAM. For the multi-streamed SSD, we used four 960GB Samsung PM963 SSD supporting eight configurable streams with modified firmware to support the multi-stream feature. Before the start of every experiment, we initialized all SSDs using NVMe command: *nvme format*. To collect the steady state I/O throughput and WAF, we measured the results after 30 minutes in all experiments. To measure I/O throughput and WAF, we used *nmon* [31] and nvme command, and *nvme smart-log*, respectively.

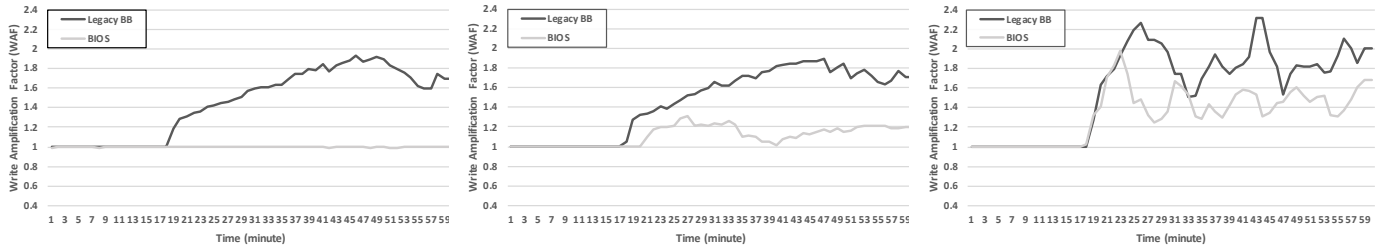
5.2. Evaluation with Synthetic Workload

In order to understand the effect of I/O separation scheme in burst buffer, we generated the synthetic workload with bursty I/O by using FIO [25] benchmark and compared that to existing burst buffer, Legacy BB, on various metrics. The synthetic workload simulates 8 users on each node, 2 threads per user; our testbed consists of 4 nodes, so the total of users is 32. Each user is assigned a different burst buffer capacity and each thread writes the 64MB file repeatedly during the runtime. If a user exceeds the allocated burst buffer capacity, 15% files are removed from the oldest one; therefore, we can consider users who are allocated a similar size of burst buffer capacity, as having a similar data lifetime.

As mentioned before, we consider shared burst buffer architecture which aggregates the bandwidth across multiple devices via data striping. We configured a shared burst buffer with four nodes having a single multi-streamed SSD. Despite using four multi-streamed SSDs supporting 8 streams, we can actually use not 32 streams but 8 streams in this burst buffer due to data striping. Each stream ID, therefore, is shared by 4 users in this experiment, thus it is important to know how users, who have different data lifetimes, are grouped into stream IDs. Depending on the degree of grouping, those users who have similar data lifetime are grouped into the same stream ID, we configured three grouping configurations which are described below.

- Configuration 1; all users in each stream ID have the same data lifetime
- Configuration 2; half of users in each stream ID have the same data lifetime
- Configuration 3; all users in each stream ID have different data lifetime

Under these configurations, we evaluated the BIOS and Legacy BB on synthetic workload for an hour, respectively. For each configuration, figure 4 illustrates the WAF sequence of Legacy BB and the BIOS. On the Legacy BB, WAF is



(a) Results for Configuration 1 (b) Results for Configuration 2 (c) Results for Configuration 3

Figure 4: WAF sequence of Legacy BB and BIOS with different configuration

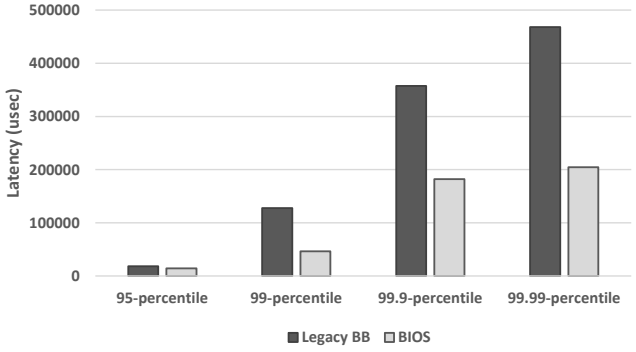


Figure 5: Comparison write tail latency between the BIOS and Legacy BB with configuration 2

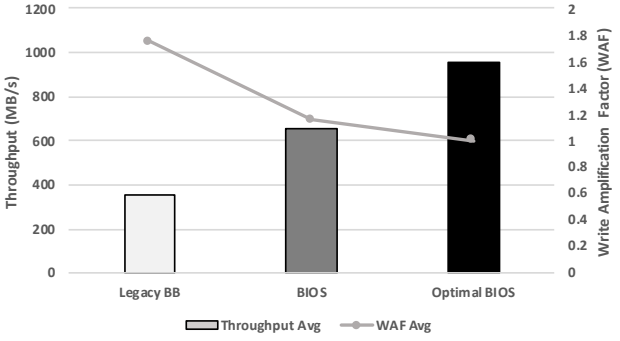
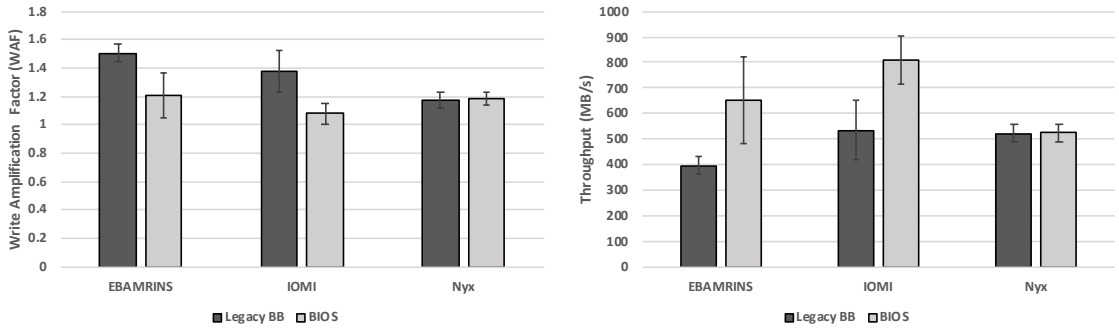


Figure 6: Results of average performance and WAF on Legacy BB, BIOS, and optimal BIOS

increased up to almost 2 times or more as time goes on regardless of grouping configurations. The meaning of data lifetime in Legacy BB is the degree of change of trend in WAF. As the randomness of the data deletion increases, the degree of copy overhead for a valid page in a block may vary when GC occurs. The Legacy BB in configuration 3 shows such a result. Actually, from the perspective of Legacy BB, grouping users who have similar data lifetime is meaningless because all data is eventually combined into the SSD block regardless of these configurations. This suggests that the Legacy BB constantly suffered GC overheads, which hurts the SSD's endurance and performance. On the contrary, the BIOS shows low and stable WAF value except for the result in configuration 3. But, even though WAF is increased by about 1.4× in configuration 3 due to 3 oversubscription for each stream, its average WAF is lower than in all case of Legacy BB because the contention of these SSD blocks is less than in the SSD blocks of Legacy BB shared by 32 users. The I/O separation scheme which decreases the degree of block sharing from multiple users in SSD offers benefits for improving the endurance in an SSD. Besides, if stream ID mapping is well optimized by considering the data lifetime, its benefits can be maximized.

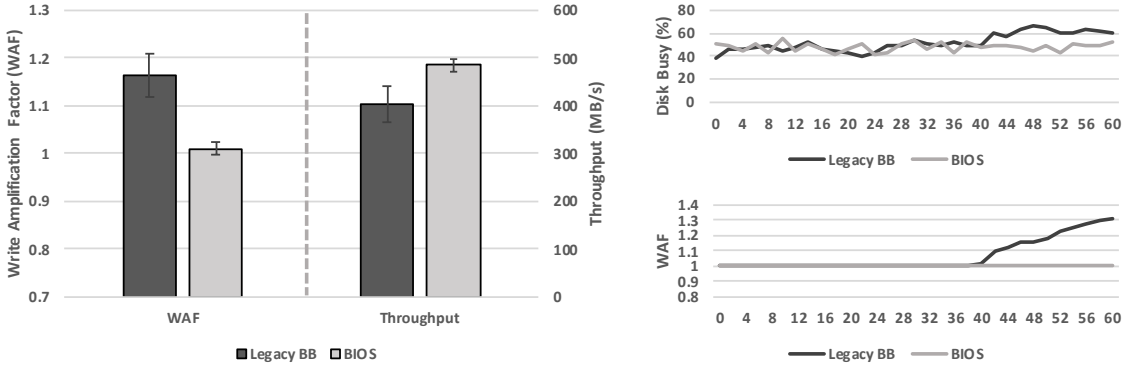
Another benefit of the BIOS is to reduce the long-tail latency. To present the effect of the I/O separation scheme for tail latency, we measured the latency for every 64MB write request in this workload. Figure 5 shows the results of latency on Legacy BB and BIOS with configuration 2. As shown in Figure 5, write requests in the BIOS become 1.27×, 2.75×, 1.96× and 2.28× faster compared to Legacy BB, at the 95^{th}, 99^{th}, 99.9^{th} and 99.99^{th} percentiles, respectively. Since the GC operation blocks the processing of incoming I/O requests until this operation is done[3], long-tail latency occurs in Legacy BB with severe GC overhead in this experiment. As a result, burst buffer with I/O separation scheme which mitigates the GC overheads provides up to 2× improved long-tail latency compared to existing burst buffer systems.

To assess the I/O separation scheme in burst buffer from a throughput point of view, we measured the throughput in these experiments performed by Legacy with configuration 2, BIOS with configuration 2, and BIOS with configuration 1 as a Legacy BB, BIOS, and optimal BIOS respectively, and calculated the average throughput after 30 minutes from the start of experiment to understand the degree of impact of GC overheads on performance. Figure 6 shows the results of average throughput and WAF from the time GC operation is fully generated in earnest. These results indicate that applying the I/O separation scheme to a conventional burst buffer can improve the endurance and performance by 1.51× and 1.83× and if stream mapping is optimized by considering data lifetime. Only then, can optimal the BIOS improve the benefits of endurance and performance up to 1.74× and 2.66× compared to Legacy BB.



(a) Average WAF (b) Average Throughput

Figure 7: Average throughput and WAF while performing experiments using HPC applications



(a) Average throughput and WAF (b) Disk busy and WAF sequence

Figure 8: Average throughput and write amplification factor when emulating supercomputing workload and disk-busy and WAF sequence of Legacy BB and BIOS during runtime

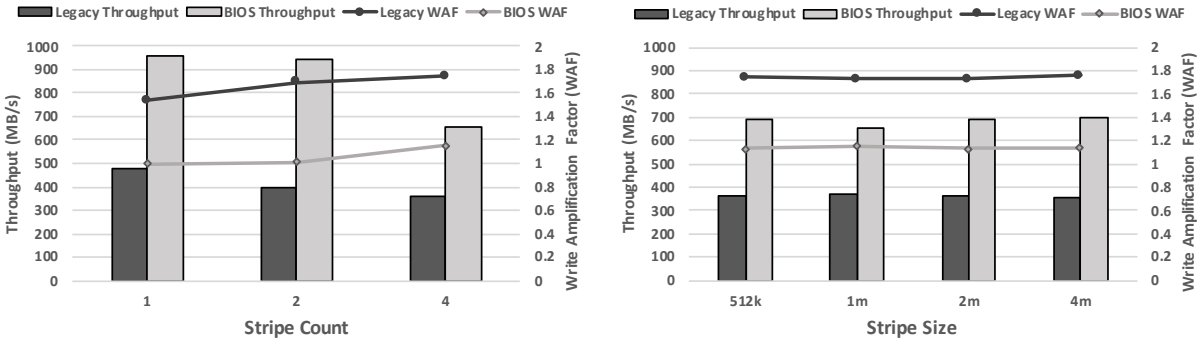
5.3. Evaluation with HPC Applications

As a next step for a more realistic evaluation, we used three HPC applications: EBAMRINS [32], IOMI [33] and Nyx [34]. Among them, EBAMRINS and IOMI are built on Chombo which is a high-performance block structured adaptive mesh refinement framework for solving partial differential equations in complex geometries. Nyx is N-body hydrodynamic cosmological simulation application that uses a massively parallel AMR code for computational cosmology. These are representative scientific applications in HPC environments. The output files generated by each application are described as follows.

- EBAMRINS : single checkpoint and plot file with 41MB and 65MB size respectively
- IOMI : single hdf5 file around 292MB size
- Nyx : checkpoint and plot directory consisting of 13 and 10 files, total size of each directory is 131MB and 117MB respectively

To load enough I/O to the burst buffer, we configured workloads to generate output files after each step of computation. We assumed the 8 users per node in total of four nodes and performed the experiments three times for each application. Therefore, a total of 32 users perform the striping I/O to the burst buffer which consists of four nodes and each stream ID is shared by 4 users in that our multi-streamed SSD supports the 8 streams. For the method of file deletion and the burst buffer capacity allocation, it is the same as for the evaluation of the synthetic workload as described in Section 5.2. In the case of EBAMRINS experiments, we used 3 threads per user in order to provide sufficient I/O load.

Figure 7 compares the BIOS with Legacy BB in terms of average WAF and write throughput on representative HPC applications. The results in EBAMRINS show that average WAF and throughput for Legacy BB are 1.5 and 395MB/s whereas WAF for BIOS are 1.2 (1.25×) and 652MB/s (1.65×) respectively. In EBAMRINS, the BIOS shows relatively higher performance deviation compared with other application's results since we used more total threads than other application experiments; stream ID is shared by more threads in EBAMRINS experiment. Therefore, depending on how well the users with similar data lifetime can be grouped together, the BIOS performance in EBAMRINS may show variable performance trends. Despite large performance deviations, the BIOS guarantees better performance than Legacy BB even in the worst case. In the IOMI application, the



(a) Experimental Results on Stripe Count

(b) Experimental Results on Stripe Size

Figure 9: Average throughput and WAF with different striping configurations

BIOS shows that 1.07 and 810MB/s for WAF and throughput respectively, whereas Legacy BB shows results of 1.37 and 534MB/s respectively. The BIOS in IOMI represents a far greater performance compared to experimental results in other applications due to the simple I/O characteristic of IOMI that generates a large single file, which allows the BIOS to keep on the benefits even though each stream ID is shared by multiple users. On the other hand, the Legacy BB results indicates that existing burst buffer can be troubled in handling the concurrent I/Os from many users many in spite of a simpler I/O pattern. Consequently, the BIOS compared to Legacy BB improves WAF and throughput by 1.27× and 1.51× on average respectively. In the case of the Nyx experiment, both burst buffers show approximately the same experimental results for WAF and throughput. In the Nyx experiment, we observed that GC overheads on the BIOS go up to the same level as Legacy BB, since the Nyx application generates multiple small files frequently and concurrently. The BIOS did not show any advantage over Legacy BB in the Nyx application. However, we expect this case to be improved in the BIOS framework. Overall, these results show that the average WAF and I/O throughput in all application experiments are improved by 1.17× and 1.37× in the BIOS when compared to the Legacy BB.

5.4. Evaluation with Emulated Workload

For a more realistic evaluation beyond the synthetic workload, we evaluated the BIOS with Darshan logs of Cori's burst buffer provided by NERSC. Darshan can profile the application's POSIX and MPI-IO function calls with minimum overheads. The Darshan log can accurately report the I/O pattern and I/O cost over the job's lifetime. More importantly, the emulated workload can represent the real I/O pattern on a production burst buffer at a national HPC facility. We selected 32 workloads which are write-intensive from the Darshan logs; the average size of write operations ranged from 1KB to 10MB. To replay these workloads, we used the workload emulator [12] and assumed that 32 users randomly select a workload and they are granted a burst buffer with a different capacity. Since the darshan log doesn't record the remove operation which is triggered by the workload manager instead of the application itself, we simply remove the data in the same way as in the synthetic tests. With the combination of different workloads for each test, we were able to reveal a different HPC workload pattern. In this subsection, we formed four combinations of workloads and conducted the experiments separately.

Figure 8a illustrates the average WAF and throughput on emulated workloads. In the case of the WAF results, the average WAF in Legacy BB is 1.16 and it is improved to 1.01 by the BIOS. Moreover, I/O throughput is improved by 1.20× with the BIOS. Even though the emulated workloads do not have enough I/O to saturate the burst buffer bandwidth, we observed that the WAF starts to increase in Legacy BB even when the disk-busy is only around 40% to 50%, as shown in Figure 8b. This phenomenon is arises from the complex I/O patterns in HPC and the concurrent I/Os from multiple users. These results indicate the BIOS prevents the degradation of performance and the device lifetime by reducing the GC overheads in such complex I/O workloads. Currently, the burst buffer has been introduced and experimentally operated in the HPC systems without being fully utilized yet[35]. However, the complete introduction and generalization of the burst buffer technology in the near future will result in more bursty I/O, and the benefits of the BIOS will become more significant.

5.5. Evaluation with Different Striping Configuration

In general, most burst buffers use the data striping which is the ability to stripe data across multiple storage devices for providing high-performance. Although striping I/O provides high-performance, it accelerates the data fragmentation in the SSD blocks, which can adversely affect the performance and lifetime of SSDs in the burst buffer. To identify the impact of data striping on the burst buffer, we performed the experiments by changing the stripe settings in regards to stripe count and stripe size. In order to give the same I/O load to each SSD regardless of the stripe count, we configured the ratio of the number of SSDs to the number of I/O users

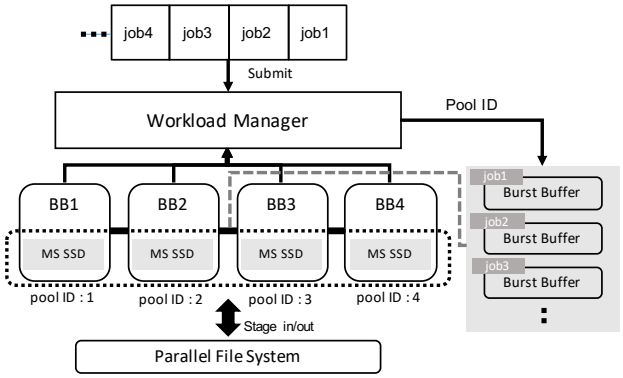
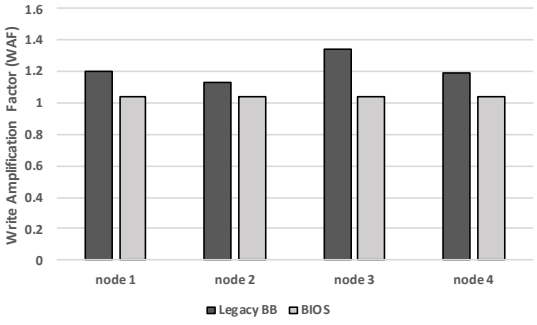
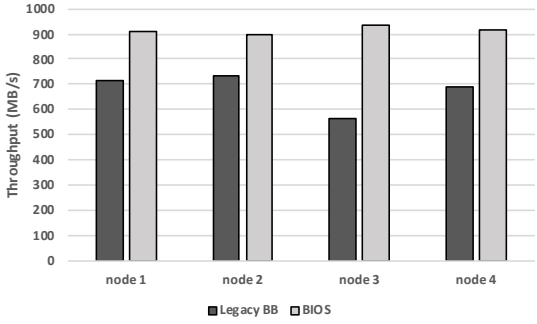


Figure 10: BIOS framework on our testbed



(a) Average WAF



(b) Average Throughput

Figure 11: Average WAF and throughput for the storage device of each node in real burst buffer environments

equally; 8 users per SSD perform I/O. In the experiment for stripe count, we set the stripe size and RDMA buffer size to the value of 512KB and 768KB separately, and for the stripe size experiment, we fixed the value of stripe count and RDMA buffer size to 4MB and 16MB respectively.

Figure 9a illustrates the average throughput and WAF while changing the stripe count. On the Legacy BB, as stripe count increases to 1,2 and 4, WAF is increased to 1.53, 1.69 and 1.74 and throughput shows that it is decreased to 477MB/s, 397MB/s and 358MB/s respectively. Each SSD handles the same amount of I/O regardless of the stripe count, but as the stripe count increases, the number of users processed by each SSD increases, resulting in more data being mixed in the SSD block. As a result, this leads to an increase of GC overheads as the stripe count increase. For the same reason, when the stripe count is 4 in the BIOS, WAF is increased up to 1.15 and its throughput is also decreased. Nevertheless, the WAF and throughput in the BIOS show 1.53×, 1.66× and 1.51×, and 2×, 2.3× and 1.83× improvement in case of 1,2 and 4 stripe count respectively.

Figure 9b shows the experimental results when the stripe size is 512KB, 1MB, 2MB and 4MB. Regardless of stripe size, the results in both Legacy BB and the BIOS show around the same WAF and throughput. Although we did not present the results for different RDMA size in this paper, we also performed the same experiment with 768KB RDMA size. But, RDMA size also did not affect performance. These results demonstrate that stripe size and RDMA size show little impact on SSD performance and lifetime in bursty I/O environments.

5.6. Evaluation on BIOS Framework

Until now, we have evaluated the BIOS directly with extensive experiments. From those experiments, we verified that the I/O separation scheme is effective and also has a limitation due to a limited number of streams. In this subsection, we confirm whether or not the BIOS framework optimizes the I/O separation scheme in burst buffer environments.

To evaluate the BIOS framework, we configured the realistic supercomputing environments using multi component workload composed of five workloads: Nyx-MiniSB, Nyx-Santabarbara, Chombo-EBAMRINS, Chombo-IOMI, and IOR benchmark[36]. We assume the 8 users each submitting the 4 jobs to the workload manager; a total 32 jobs are submitted and running on the BIOS framework together while performing the IOR applications with large files in order to fill the burst buffer capacity. As shown in Figure 10, we built the BIOS framework on our testbed which has four nodes. Due to the limited number of nodes in our testbed, we configured the four burst buffer pools using four SSDs; each burst buffer pool consists of single SSD, so each job only uses a single SSD in this framework. For example, when a job is submitted to the workload manager, the BIOS framework assigns the burst buffer pool ID based on stream-aware scheduling policy and provides burst buffer using the single SSD in the allocated pool ID.

Figure 11 illustrates the average WAF and throughput for all SSDs in the framework. In Figure 11a and 11b, the WAF and throughput in BIOS show 1.04 and 914MB/s on average while presenting the 1.21 and 675MB/s in Legacy BB respectively. The BIOS compared with Legacy BB improves WAF and throughput by 1.17× and 1.35× respectively, which indicates that the BIOS framework effectively works as designed in realistic supercomputing environments. Under the BIOS framework, each job can exclusively use the SSD without interference from other jobs, so the BIOS removes the GC overheads in the SSD despite operating with a lot of jobs, providing consistent benefits of I/O separation scheme.

In Figure 11, the performance results show slight differences between nodes. This is because each node handles a different combination of applications due to the pool ID

allocation by the workload manager. In addition, an interesting finding is that the WAF in the BIOS is not 1.00 but 1.04 even though each job uses its own stream ID. 0.04% copy overheads come from the different timing which marks the invalid state about deleted data between the file system and an SSD. When a user deletes the data, the file system marks data block as not in use, but the SSD does not know which SSD's page should be marked as invalid until the operating system notifies it. This results in unnecessary copy operation for worthless data, and it causes the GC overheads despite using the SSD completely alone. However, such a degree of overheads is negligible and can be completely removed by using trim command [37].

6. Summary and lessons learned

In this section, we summarize the findings and insights from an extensive evaluation performed in the burst buffer with an I/O separation scheme and the BIOS framework.

6.1. An I/O Separation Scheme in Burst Buffer

6.1.1. Evaluation with Synthetic Workload

Applying I/O separation scheme to the burst buffer can mitigate garbage collection overheads efficiently, improving I/O throughput, device lifetime, and service level objective. When data grouping is precisely grouped according to data lifetime, the benefits of I/O separation scheme are maximized.

6.1.2. Evaluation with HPC Applications

In real HPC applications with diverse I/O patterns, the I/O separation scheme demonstrates that GC overheads are mitigated in SSDs of the burst buffer. Although a burst buffer with an I/O separation scheme provides inconsistent benefits depending on the I/O pattern, it ensures better, overall performance than do existing burst buffers.

6.1.3. Evaluation with Emulated Workload

Currently, the I/O load in real supercomputing workloads is not enough to saturate the SSDs of the burst buffer, representing 40% to 50% disk-busy since burst buffers have been introduced and experimentally operated in HPC systems. Despite insufficient loads, the complex I/O patterns bring the GC overheads in existing burst buffers, whereas the BIOS completely eliminates the GC overheads incurred by complex I/O patterns.

6.1.4. Evaluation with Striping Configurations

The striping I/O pattern used in the shared burst buffer accelerates the data fragmentation in SSD blocks, undermining the benefits of the I/O separation scheme. Management is required to maintain on the benefits of the I/O separation scheme in burst buffers. Without addressing the problem of GC overheads in SSD, trivial optimization such as stripe size or RDMA buffer size is negligible in optimizing the I/O separation scheme in burst buffer environments.

6.2. A BIOS Framework

6.2.1. Evaluation with Real Burst Buffer Environments

The real burst buffer system, the BIOS framework, demonstrates that the I/O separation scheme can be optimized in burst buffer environments through the burst buffer pools and stream-aware scheduling policy. The burst buffer pools reduce interference caused by data striping. The stream-aware scheduling policy solves the skewed stream allocation for jobs of the same user while balancing the load. The BIOS framework can reduce the GC overhead which is no longer reduced by limitations from the BIOS.

7. Discussion

7.1. Limited Number of Nodes

In this paper, even though we conducted the evaluation of the BIOS framework on a small-scale cluster consisting of a limited number of nodes, the effectiveness of BIOS can be expanded to large-scale clusters. Because the function of the BIOS framework is to manage the burst buffer resource by burst buffer pools, it can be adapted to any size cluster by adjusting the size of the burst buffer pools. For example, if we assume the scaled-up cluster with 16 burst buffer nodes, the BIOS framework will look like Figure 3. We can configure the four burst buffer pool IDs grouped into four SSDs considering the scale of the system. Assuming multiple jobs are running in same pool ID in this cluster, we can consider this pool ID to be equivalent to the same circumstances in the BIOS experiments of Section 5.2 to 5.4, because these experiments are performed on four nodes using the striping I/O. Although we use a limited number of nodes to demonstrate the effect of BIOS framework, its benefits can be generalized to a large-scale cluster.

Best above all, using multi-stream with burst buffers in a supercomputing system requires collaboration from device builders and system designers.

7.2. Advanced BIOS Framework

In this paper, we configured the fixed number of storage device for each burst buffer pool ID. The management of the BIOS in uniform pool IDs allows us to make it easier to allocating burst buffer resources and to ensure a certain level of performance. However, as a demand of the I/O requirement of each HPC application varies, the framework with uniform pool IDs can lose the chance to be used more efficiently such as grouping the I/O streams with similar characteristic into pool IDs. Therefore, by organizing the pool IDs with a variable number of SSDs, we expect our framework will manage storage more efficiently and further improve overall system performance.

We can also utilize the information of a job's lifetime in the workload manager in order to predict when the data of the job will actually be erased. If we use this information well, we can ideally group the data into the stream IDs; Theoretically, all data in the same stream will be erased at the same time, which will show the same BIOS results shown in Figure 4a.

We expect that utilizing the job's lifetime information enable users to make BIOS more effective.

8. Related work

As a beginning of the burst buffer study, Liu *et al.* explored the potential of burst buffer and demonstrated its effectiveness in a large-scale HPC system. In subsequent studies, the study for handling the I/O bottleneck problem in HPC systems[1, 7], the study of new burst buffer architecture[8, 9] and the study of scheduling policy for I/O between burst buffer and PFSs, or burst buffer resource[6, 10, 11] have continued progress. Our work is based on the burst buffer architecture represented in Bhimji *et al.*[1], and we improve the performance and endurance problem that might arise in this system by mitigating the GC overheads.

The multi-streamed feature has been introduced for mitigating the GC overheads of SSD. A large body of prior research has been conducted on how to effectively leverage this multi-stream mechanism. There are some strategies to leverage the multi-stream feature, which typically includes mapping data from the applications level [14], [38], file systems layer [23], and the block layer [15]. In case of application level customization, the multi-stream feature can be optimized via the understanding of data lifetime of the application although it comes at a cost; a compatibility issue to all applications requiring the multi-stream feature. The others using abstracted information support multi-stream feature with transparency to applications but have limitations in optimizing all applications compared to application level customization. A recent study by Kim *et al.* has proposed the automatic stream management based on application level information, program context. Our work also provides automatic stream management in burst buffer by utilizing intuitive and effective data lifetime information based on burst buffer I/O characteristics.

9. Conclusions

With emerging burst buffers, the HPC systems with disk-based PFSs could satisfy the I/O requirement. However, the burst buffers also cannot completely meet the I/O requirement since some I/O characteristics of the HPC environment may cause the write amplification in SSDs of the burst buffer, leading to the performance degradation of the entire burst buffer system. To address this problem, we have proposed the BIOS, a Burst Buffer with an I/O separation scheme based on multi-streamed SSDs and a framework managing the BIOS efficiently in burst buffer environments. By assigning the stream to each user transparently, the BIOS provides the illusion that each user is using their own SSDs; actually, each user uses exclusive NAND blocks in the same SSD, this mitigates the write amplification in the SSDs, possibly enhancing the performance by an average of 1.44× and reducing WAF by 1.20×, as shown in our extensive experiments. In addition, the framework manages the BIOS using stream-aware scheduling policy based on burst buffer pools, optimizing the I/O separation scheme regardless of cluster scale. Although our experimental setup uses 4 nodes due to lack of multi-streamed SSDs available on current burst buffer implementation, our diverse experiments and scalable feature of the BIOS framework let us uncover and analyze the benefits and limitations of the I/O separation scheme in the real burst buffer environments. Therefore, the results of all those tests demonstrate a very promising future of using the BIOS framework in HPC environments. For future studies, we plan to enhance our framework by improving the stage in/out using dcp[39] or gnu parallel[40] utility for large files. In addition, to support this transparent viewing, we will ultimately implement the BIOS framework which supports the transparent caching mode.

10. Acknowledgment

This work is supported by the Korea Institute of Science and Technology Information (Grant No.K-20-L02-C08), and Next-Generation Information Computing Development Program(NRF-2015M3C4A7065646) and Basic Science Research Program(NRF-2020R1F1A1072696) through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT. It was also supported by National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP)(NRF-2016M3C4A7952587, NRF-2017R1A2B4004513) and BK21 FOUR Intelligence Computing(Dept. of Computer Science and Engineering, SNU) funded by National Research Foundation of Korea(NRF) (4199990214639). And this work was supported by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, and also used resources of the National Energy Research Scientific Computing Center (NERSC).

References

[1] 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.

[2] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, 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.

[3] S. Yan, H. Li, M. Hao, M. H. Tong, S. Sundararaman, A. A. Chien, H. S. Gunawi, Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in nand ssds, ACM Transactions on Storage (TOS) 13 (3) (2017) 22.

[4] Y. Luo, Y. Cai, S. Ghose, J. Choi, O. Mutlu, Warm: Improving nand flash memory lifetime with write-hotness aware retention management, in: 2015 31st Symposium on Mass Storage Systems and Technologies (MSST), IEEE, 2015, pp. 1–14.

[5] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, W. Allcock, I/o performance challenges at leadership scale, in: High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on, IEEE, 2009, pp. 1–12.

[6] S. Thapaliya, P. Bangalore, J. Lofstead, K. Mohror, A. Moody, Managing i/o interference in a shared burst buffer system, in: 2016 45th International Conference on Parallel Processing (ICPP), IEEE, 2016, pp. 416–425.

[7] J. Lofstead, I. Jimenez, C. Maltzahn, Q. Koziol, J. Bent, E. Barton, Daos and friends: a proposal for an exascale storage system, in: SC'16:

Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2016, pp. 585–596.

[8] J. Bent, S. Faibish, J. Ahrens, G. Grider, J. Patchett, P. Tzelnic, J. Woodring, Jitter-free co-processing on a prototype exascale storage stack, in: 012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), IEEE, 2012, pp. 1–5.

[9] T. Wang, W. Yu, K. Sato, A. Moody, K. Mohror, Burstfs: A distributed burst buffer file system for scientific applications, Tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States) (2016).

[10] K. Tang, P. Huang, X. He, T. Lu, S. S. Vazhkudai, D. Tiwari, Toward managing hpc burst buffers effectively: Draining strategy to regulate bursty i/o behavior, in: 2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2017, pp. 87–98.

[11] W. Liang, Y. Chen, J. Liu, H. An, Contention-aware resource scheduling for burst buffer systems, in: Proceedings of the 47th International Conference on Parallel Processing Companion, ACM, 2018, p. 32.

[12] J. Han, D. Koo, G. K. Lockwood, J. Lee, H. Eom, S. Hwang, Accelerating a burst buffer via user-level i/o isolation, in: 2017 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2017, pp. 245–255.

[13] J.-U. Kang, J. Hyun, H. Maeng, S. Cho, The multi-streamed solid-state drive., in: HotStorage, 2014.

[14] F. Yang, K. Dou, S. Chen, M. Hou, J.-U. Kang, S. Cho, Optimizing nosql db on flash: A case study of rocksdb, in: Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), 2015 IEEE 12th Intl Conf on, IEEE, 2015, pp. 1062–1069.

[15] J. Yang, R. Pandurangan, C. Choi, V. Balakrishnan, Autostream: automatic stream management for multi-streamed ssds, in: Proceedings of the 10th ACM International Systems and Storage Conference, ACM, 2017, p. 3.

[16] C. S. Daley, D. Ghoshal, G. K. Lockwood, S. Dosanjh, L. Ramakrishnan, N. J. Wright, Performance characterization of scientific workflows for the optimal use of burst buffers, Future Generation Computer Systems.

[17] D. Henseler, B. Landsteiner, D. Petesch, C. Wright, N. J. Wright, Architecture and design of cray datawarp, Cray User Group CUG.

[18] W. Schenck, S. El Sayed, M. Foszczynski, W. Homberg, D. Pleiter, Early evaluation of the âĂIJinfinite memory engineâĂİ burst buffer solution, in: International Conference on High Performance Computing, Springer, 2016, pp. 604–615.

[19] Slurm: A Highly Scalable Workload Manager, https://github.com/SchedMD/slurm (2018).

[20] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, 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.

[21] A. Huffman, Nvm express revision 1.3 specification, NVM Express, Inc.

[22] T. Kim, D. Hong, S. S. Hahn, M. Chun, S. Lee, J. Hwang, J. Lee, J. Kim, Fully automatic stream management for multi-streamed ssds using program contexts, in: 17th {USENIX} Conference on File and Storage Technologies ({FAST} 19), 2019, pp. 295–308.

[23] E. Rho, K. Joshi, S.-U. Shin, N. J. Shetty, J.-Y. Hwang, S. Cho, D. D. Lee, J. Jeong, Fstream: managing flash streams in the file system, in: 16th USENIX Conference on File and Storage Technologies, 2018, p. 257.

[24] T. Kim, S. S. Hahn, S. Lee, J. Hwang, J. Lee, J. Kim, Pcstream: automatic stream allocation using program contexts, in: 10th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 18), 2018.

[25] Flexible I/O Tester, https://github.com/axboe/fio (2018).

[26] D. U. Tony Wildish, Using the Burst Buffer on Cori, https://www.nersc.gov/assets/Uploads/Burst-Buffer-tutorial.pdf (June 2017).

[27] BeeGFS Documentation, https://www.beegfs.io/wiki/TableOfContents (2018).

[28] BeeGFS Storage Pools, https://www.beegfs.io/wiki/StoragePools (2018).

[29] H. Khetawat, C. Zimmer, F. Mueller, S. Atchley, S. S. Vazhkudai, M. Mubarak, Evaluating burst buffer placement in hpc systems, in: 2019 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2019, pp. 1–11.

[30] How to use the Burst Buffer, https://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/ (2019).

[31] N. Griffiths, nmon performance: A free tool to analyze aix and linux performance (2003).

[32] G. Miller, D. Trebotich, An embedded boundary method for the navier–stokes equations on a time-dependent domain, Communications in Applied Mathematics and Computational Science 7 (1) (2011) 1–31.

[33] D. Devendran, S. Byna, B. Dong, B. Van Straalen, H. Johansen, N. Keen, N. F. Samatova, Collective i/o optimizations for adaptive mesh refinement data writes on lustre file system (2016).

[34] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukić, E. Van Andel, Nyx: A massively parallel amr code for computational cosmology, The Astrophysical Journal 765 (1) (2013) 39.

[35] Burst Buffer Early User Program, http://www.nersc.gov/users/computational-systems/cori/burst-buffer/burst-buffer-early-user-program/ (January 2017).

[36] Parallel filesystem I/O benchmark, https://github.com/LLNL/ior (2012).

[37] J. Kim, H. Kim, S. Lee, Y. Won, Ftl design for trim command, in: The Fifth International Workshop on Software Support for Portable Storage, 2010, pp. 7–12.

[38] T.-D. Nguyen, S.-W. Lee, Optimizing mongodb using multi-streamed ssd, in: Proceedings of the 7th International Conference on Emerging Databases, Springer, 2018, pp. 1–13.

[39] distributed file copy program, https://github.com/hpc/dcp (2012).

[40] O. Tange, GNU Parallel: The Command-Line Power Tool, https://www.usenix.org/system/files/login/articles/105438-Tange.pdf (February 2011).

Donghun Koo received his B.S in Electronics and Information Engineering from Korea Aerospace University, and M.S. in Computer Science from Seoul National University. He is a researcher in TmaxOS, Korea. His research interests are Graphics Kernel, storage system, distributed system and high-performance computing.

Jaehwan Lee is an assistant Professor in the Department of Electronics and Information Engineering at Korea Aerospace University. He received his B.S. and M.S. in Electrical Engineering from Seoul National University, and Ph.D. in Computer Science from University of Maryland at College Park. He has several industry research experiences; Korea Telecom (KT) as a senior researcher (2000 2005), NEC labs in America and Bell labs, Alcatel-lucent as a research intern, and Samsung System Architecture lab in US as a Research Staff Engineer. His research interests include distributed computing, high-performance computing, and Big-data infrastructures to support data intelligence. He was a recipient of the General Electric (GE) Scholarship and the Korean Government Scholarship for Electric Power Industry.

Jialin is a computer engineer in the Data Analytics Service group. He received his Ph.D. in computer science from Texas Tech University in 2015. He spent the summer of 2013 with the SDM group in building 50 working on scientific data service (SDS). He has interest in scientific data management/analytics and parallel I/O.

Eun-Kyu Byun received his B.S. and Ph. D. degrees in computer science from Korea Advanced Institute of Science and Technology(KAIST), in 2003 and 2011, respectively. He is a senior researcher in Korea institute of science and technology information(KISTI). Before joining KISTI, he worked in SK Telecom as a senior research staff from 2011 to 2013. His research interests include distributed system, high performance computing and storage system.

Jae-Hyuck Kwak is a senior researcher in Korea Institute of Science and Technology Information (KISTI). After joining KISTI, He has been involved in various HPC projects at Supercomputing Center; from infrastructure building, middleware development to international collaboration. He received his M.S. in School of Computer Science and Engineering from Seoul National University. His research area includes High Performance computing, Distributed Computing and Big Data Computing, and especially in the convergence of AI / Big Data and HPC.

Glenn K. Lockwood is a storage architect who specializes in I/O performance analysis, extreme-scale storage architectures, and emerging I/O technologies. His research interests revolve around understanding I/O performance by correlating performance analysis across all levels of the I/O subsystem, from node-local page cache to backend storage devices. To this end, he is actively involved in the development of TOKIO, a framework for holistic I/O performance characterization. He is also a maintainer of the IOR and mdtest community I/O benchmarks and contributes to the Darshan I/O profiling library.

Soonwook Hwang received his B.S. in Mathematics, M.S. in Computer Science and Statistics from Seoul National University, Korea, Ph.D. in Computer Science from University of Southern California, USA. He was a Researcher in the National Institute of Informatics, Japan. He is currently a Principle Researcher in the National Institute of Supercomputing and Networking at KISTI (Korea Institute of Science and Technology Informations). His research interests are in Grid and Clould Computing, and High Performance Distributed Computing.

Katie is the NERSC Division Deputy and leads the Data Department at NERSC. As the Data Department Head at the National Energy Research Scientific Computing (NERSC) Center, Katie has oversight of the Data Science Engagement, Data and Analytics Services, Storage Systems, and Infrastructure Services groups. Katie is the Project Director for the NERSC-9 supercomputing project, a project to deploy NERSC's next supercomputer in 2020. The system is expected to be announced in the fall of 2018. She is also the PI on a ASCR funded research project called ScienceSearch: Enabling Automated Metadata through Machine Learning. Katie has expertise in system architectures, parallel I/O, application performance and user science requirements.

Kesheng Wu works actively on a number of topics in data management, data analysis, and high-performance computing. His algorithmic research work includes statistical methods for feature extraction, indexing techniques for searching large datasets, and matrix based techniques for machine learning and scientific computing. He has authored and coauthored more than 100 technical publications, 15 of which have more than 100 citations each. Kesheng received a Ph.D. in computer science from the University of Minnesota, and a B.S. in physics from Nanjing University.

Hyeonsang Eom received the BS degree in computer science and statistics from Seoul National University (SNU), Seoul, Korea, in 1992, and the MS and PhD degrees in computer science from the University of Maryland at College Park, Maryland, USA, in 1996 and 2003, respectively. He is currently a full professor in the Department of Computer Science and Engineering at SNU, where he

has been a faculty member since 2005. He was an intern in the data engineering group at Sun Microsystems, California, USA, in 1997, and a senior engineer in the Telecommunication R&D Center at Samsung Electronics, Korea, from 2003 to 2004. His research interests include distributed systems, cloud computing, operating systems, high performance storage systems, energy efficient systems, fault-tolerant systems, security, and information dynamics.