2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

Interference-Aware I/O Scheduling for Data-Intensive Applications on Hierarchical HPC Storage Systems

Weihao Liang*, Yong Chen†, Hong An*
* School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
† Department of Computer Science, Texas Tech University, Lubbock, TX, United States
Email: lwh1990@mail.ustc.edu.cn, yong.chen@ttu.edu, han@ustc.edu.cn

***Abstract*—Scientific applications in critical areas are becoming more and more data-intensive. As data volume continues to grow, the data movement from memory to storage system has turned into a crucial performance bottleneck for many data-intensive applications. Newly emerged burst buffer concept provides a promising solution by increasing the depth of storage hierarchy to increase I/O performance of data-intensive applications. However, the data management on such multi-layer hierarchical storage system is still understudied. How to leverage each layer of storage for efficient data movement is an important research topic for HPC field. In this paper, we present a dynamic, interference-aware scheduling scheme that can efficiently manages the I/O scheduling among different layers of hierarchical HPC storage system to coordinate multiple concurrent data-intensive applications. Extensive experiments have been conducted and the results have demonstrated that our proposed approach can significantly improve the I/O performance of data-intensive applications.**

***Index Terms*—Burst Buffer, Data Movement, Storage System, I/O Scheduling, I/O Interference**

I. Introduction

It is well acknowledged that many scientific applications in critical areas, such as climate science, astrophysics, combustion science, computational biology, and high-energy physics, have become highly data intensive and pose critical big data challenges to the research and development community [1], [2]. Large amounts of data are generated from scientific experiments, observations, and simulations. The volume of these datasets can range from hundreds of gigabytes to hundreds of petabytes or even beyond. For instance, the FAST project (500-meter Aperture Spherical Telescope) [3] in Australia uses the Next Generation Archive System (NGAS) to store and maintain a large amount of data collected. NGAS expects to handle about 3 petabytes of data from FAST every year, enough to fill 12,000 single-layer, 250 gigabyte Blu-ray disks. There are a number of reasons for this big data revolution: a rapid growth in computing capability (especially when compared with a much slower increase in I/O system bandwidth) has made data acquisition and generation much easier; high-resolution, multi-model scientific discovery will require and produce much more data; and the needs that insights can be mined out of large amounts of low-entropy data have substantially increased over years.

In such a big data era, how to efficiently process and analyze rapidly growing scientific datasets is a key challenge. Due to the enlarging gap between the rapidly increasing computing ability and the slowly improving I/O bandwidth, data access has become the critical performance bottleneck of many scientific applications which usually contain a large number of I/O accesses where large amounts of data are written to and retrieved from the storage system. Newly emerged burst buffer [4] concept provides a promising solution for bridging the I/O gap by adding layers with fast storage medium (e.g. SSDs or NVMs). For example, Cori [5] at the National Energy Research Scientific Computing Center (NERSC) and Trinity [6] at Los Alamos National Laboratory (LANL) used SSD-based dedicated nodes to quickly absorb bursty I/O traffic. Summit [7] at Oak Ridge National Lab (ORNL) and Sierra [8] at Lawrence Livermore National Laboratory (LLNL) provision local NVMe-device in each compute node for fast buffering. The projected upcoming exascale supercomputer [9] to be released around 2021 expects to be equipped with multiple heterogeneous, high-speed storage layers in the I/O subsystem. Arguably it becomes a critical challenge how to utilize all these storage layers efficiently to well serve data-intensive applications. With more high-speed storage layers introduced into storage system, data management among different layers has become more complicated than ever before and can be a burden for users or programmers. Furthermore, when many data-intensive applications access such hierarchical storage system concurrently, the complexity of data movement and interaction can be particularly difficult to handle. Therefore, there is a strong desire to have a transparent and automated data movement scheduling support for such deep, multi-layer hierarchical storage system.

In this paper, we propose an interference-aware scheduling (IAS) scheme that manages the data movement among different layers of hierarchical HPC storage system to coordinate concurrent accesses from data-intensive applications. By considering the status of applications' I/O activities and each layer of the storage system at the runtime, IAS can make decisions for scheduling data movement to reduce I/O interference among different applications and thus improve the overall performance. By placing data among multiple storage layers automatically, such an efficient, automated data scheduling approach provides a promising, reliable solution for the I/O challenges towards the hierarchical storage architecture

978-1-7281-2058-4/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2019.00099

of future exascale HPC system.

The contributions of this research study include:

- we present the design and implementation of a new I/O scheduler for multi-layer storage system, namely an interference-aware scheduling (IAS) scheme.
- we introduce a novel data movement strategy to explore potentials of all layers in the hierarchical HPC storage system.
- we conduct extensive experiments and the results have demonstrated that the proposed approach can significantly improve the I/O performance of data-intensive applications.

The rest of paper is organized as follows. Section II discusses background and motivation of this research. Section III presents the design of the proposed interference-aware I/O scheduling approach. Experimental and analytical results are presented in Section IV. Section V discusses the existing work and compares this study with them, and Section VI concludes this study and discusses possible future work.

II. Background and Motivation

A. Hierarchical HPC Storage Systems

The typical I/O patterns of data-intensive scientific applications running on HPC system are mostly bursty, periodic I/O phases that frequently occur between computation phases [10]. For better management of these I/O phases from data-intensive applications, multiple new storage layers have been introduced between the applications and the remote parallel file system. We call it hierarchical HPC storage systems. The architecture overview of hierarchical HPC storage systems can be seen in Figure 1. Apart from the memory (RAM), extra non-volatile memory (NVM) device is installed in each compute node that provides high bandwidth local storage and significantly reduces the time for file I/O. For example, it allows checkpoints to be written locally and discarded once the next checkpoint is completed [11]. In addition, the burst buffer resides on dedicated nodes that are usually deployed with SSDs or as a high-speed storage tier, will most likely be present and connected closely to the compute nodes with high speed network such as Infiniband. As shared resource, the burst buffer nodes are available for all compute nodes to access directly. While applications running in compute nodes issue bursty I/O operations, the burst buffer nodes can quickly absorb I/O traffics in their local SSDs, and let the applications continue to the next computing phase as soon as possible. The data temporarily stored in the shared burst buffers can then be transferred to the remote permanent storage nodes (parallel file system, PFS) asynchronously without interrupting the applications.

B. Motivation and Challenges

Scientific applications are becoming more and more data-intensive. They are always required to process large amounts of data and show bursty, periodical I/O pattern such as checkpointing which frequently appear between computation phases. In traditional HPC storage system, many applications

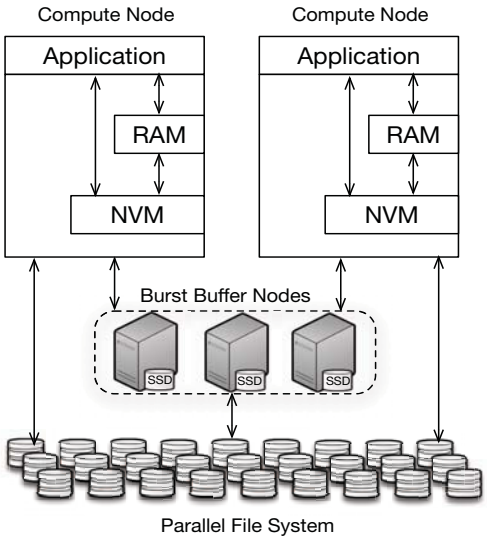

Fig. 1. Overview of Storage Hierarchy in HPC Systems.

spend most of the entire execution time in moving the data from compute nodes to storage nodes, exposing significant performance bottleneck on data-intensive applications. The hierarchical HPC storage system offers a potential solution by adding more storage layers between compute nodes and storage nodes that can accelerate scientific applications in many cases, such as fast checkpoint/restart, asynchronous data flushing and in-transit data analysis. However, while more and more heterogeneous storage layers are adding to HPC storage system, how to efficiently utilize all these storage layers to serve well data-intensive applications is crucial, posing significant challenges on system software design. These challenges are two-fold, as discussed in detail below.

First, traditional I/O libraries for processing scientific dataset only provide end-to-end I/O operations, which lack awareness of intermediate layers in such hierarchical storage system. In this case, the responsibility of using these high-speed storage layers has left to users which may become a severe burden on them. As the storage hardware is developing very fast, it is unrealistic to modify the source codes of plenty of legacy scientific applications to adapt every new storage architecture. System software needs a transparent solution (e.g. middleware) to manage these multiple intermediate layers to gain high I/O performance and hide the architecture complexity from users.

Second, each storage layer is an isolated subsystem and data management among different layers are becoming more complicated with increasing number of layers. Therefore, new data movement and flushing strategies are required to support this multi-layer architecture and make data automatically flowing between different layers. In addition, as HPC system is a public shared compute and storage resource,

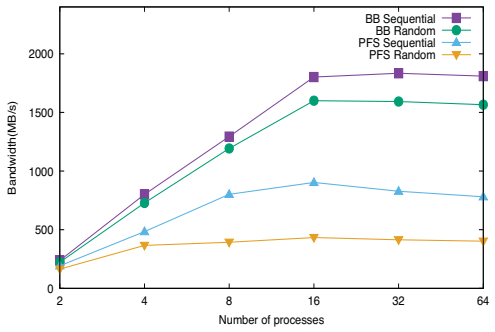

Fig. 2. I/O bandwidth variation with different I/O patterns on different storage layers

when more and more data-intensive applications with different I/O pattern simultaneously accessing to storage system, it is important to realize the potential I/O interference may happen on certain storage layer which may lead to performance loss. To understand the relationship between I/O concurrency and performance on different storage layers, we used IOR [4] to conduct a series of experiments. Two different I/O patterns were tested: contiguous and random which indicate application issues sequential I/O requests and random I/O requests. We use 4 I/O nodes equipped with HDD as parallel file system and other 4 nodes equipped with SSD as burst buffers. The number of total processes varied from 2 to 64. All these processes were simultaneously accessing to burst buffer and PFS respectively. The experimental results are presented in Figure 2. It can be observed that the bandwidth increased first and then dropped with the increase of the number of processes in both contiguous and random pattern cases. The reason for the bandwidth increase is that few I/O processes cannot make burst buffer and PFS fully utilized. After keeping increasing the number of processes, the I/O throughput began to saturate or drop slightly. The reason for the throughput degradation was caused by the I/O interference from too many I/O processes concurrently accessing the same shared storage layer. If more applications or processes are accessing to certain one layer in the same time, it will make the performance loss even worse. In addition, we can observe that the burst buffer had smaller degree of bandwidth degradation and much higher I/O bandwidth than PFS. It is because the SSD is more desired for high throughput and low delay of random access than HDD. In conclusion, an efficient data movement strategy should be aware of structure differences and avoid I/O interference from cross-application.

III. Methodology

A. Framework Overview

In order to efficiently and transparently manage data movement across multiple layers of hierarchical HPC storage system, we propose IAS, an Interference-Aware I/O Scheduling framework. The framework is designed as a middle-ware layer resided between applications and underlying hierarchical storage systems including local non-volatile memory (NVM) device, dedicated shared burst buffer nodes (SSDs) and shared, global parallel file system (HDDs). As a middle-ware design, it intends to intercept I/O operations from applications and move data to different destinations according to various data scheduling strategies. The IAS framework design is friendly and compatible to existing scientific applications. Our objective is to improve both the application performance and the system efficiency by coordinating I/O requests from different applications to reduce cross-application interference and maximize the utilization of each storage layer. Furthermore, IAS also provides options for users to select their preferred scheduling strategies and implement a user-guided scheduling scheme to better manage the storage resources.

B. Design and Implementation

Figure 3 shows a high-level view of the overall IAS framework. It provides a transparent and efficient data management solution for the hierarchical HPC storage system. We describe each component below.

Runtime. The runtime monitors the status of computation and I/O phases of all active applications. The runtime runs in the background on each compute node, communicates with the metadata organizer and data scheduler. When each application enters or finishes a certain I/O phase, the runtime will intercept I/O calls by capturing the time stamp and send message to data scheduler to inform the begin or end of each I/O phases. For example, When an application starts and calls like write() on each of its nodes, the runtime intercepts this call and sends a signal to the runtime to indicate that the application has started an I/O operation on that compute node. The signal also includes information about the application such as number of I/O process. Communication between the scheduler and runtime has negligible overhead because messaging occurs in the background and thus does not directly affect application execution time.

Metadata Organizer. The metadata organizer is responsible for keeping track of metadata operations (i.e., file name, permissions etc.) of every data file during I/O phases. When an I/O request has been processed by data scheduler, the complete data file may be divided to multiple data chunk and chunks are buffered in different storage layer for next movement. So metadata organizer is also responsible for monitoring the placement of all buffered data chunks to keep a complete view of data file for applications. Additionally, metadata organizer is also responsible for maintaining the status of each storage layer such as available space and I/O concurrency (i.e. the number of concurrent running applications).

Data Scheduler. Data scheduler is designed for automated data movement across all the storage layers. In order to achieve high I/O performance for applications, data scheduler detects the availability of each storage layer and analyze each application's I/O state to make data scheduling decisions. For example, if an application is ready to issue an I/O request, the data scheduler retrieves the information of the I/O request from

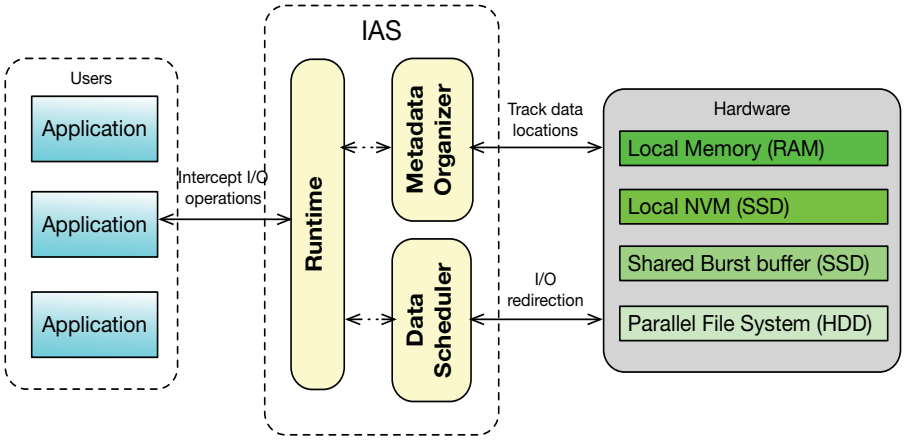

Fig. 3. IAS Framework Overview.

the runtime and status of each storage layer from metadata organizer to determine the next buffering destination for this I/O request by applying the selected data scheduling algorithms. The data scheduler is customizable to deploy different data scheduling strategies or algorithms. And the decisions of when or where to flush the buffered data is also based on the selected scheduling strategies. Its goal is to minimize I/O interference between different applications simultaneously accessing the same shared storage layer. In addition, the data scheduler is also responsible for the date flushing from shared burst buffer nodes to parallel file system.

C. Scheduling Strategies

In this section, we present three types of I/O scheduling strategies for coordinating multiple data-intensive application concurrently accessing the hierarchical storage system. Baseline is the default way to perform I/O operations in the storage system that only have hdd-based parallel file system. By contrast, high-level priority and interference-aware are scheduling algorithms in which have ssd-based burst buffer shared by all compute nodes, thus increasing the opportunity for scheduling of data redirection. By managing different layers of hierarchical HPC storage system, IAS can provide the global view of running applications with concurrent accesses to the underlying hierarchical storage resource and prevent I/O interference in each level by employing the interference-aware scheduling strategies. Each strategy is described in detail below.

1) Baseline: Figure 4(a) outlines the baseline scheduling scheme, it only accesses the PFS when an application is entering I/O phase. At the start of an I/O phase, the application just forward its data into the PFS. I/O interference can happen when multiple applications access to PFS in the same time, leading to further degradation of application performance as a result of I/O contention. The baseline scheme represents the conventional end-to-end I/O procedure in the traditional HPC storage system.

2) High-level Priority (HP): The goal of this strategy is to leverage the high bandwidth of burst buffers for applications when accessing the hierarchical storage system. In the High-level Priority (HP) scheduling strategy, data scheduler can utilize ssd-based burst buffer which are shared by multiple applications and provide much higher bandwidth than PFS. As shown in Figure 4(b), when each application enters an I/O phase, the data are first buffered in the shared burst buffer and return to compute phase quickly. And the burst buffer nodes can asynchronously flush the data into the PFS. This strategy is similar to the default data buffering policy of Cray's DataWarp [12] for the burst buffer subsystem in Cori supercomputer [5]. However, concurrently accessing to the shared burst buffer will cause the I/O contention problem [13] which may also cause I/O performance degradation of applications.

3) Interference-Aware (IA): The goal of this strategy is to both improve all storage layers' utilization and reduce I/O interference from cross-application in the shared storage layers (i.e. shared burst buffer and PFS). Figure 4(c) illustrates a common concurrent access example for the interferene-aware strategy, when *App1* performs I/O request, it first buffers its data in the shared burst buffer and then return to computing to let burst buffer nodes handle asynchronous data flushing. After some time if another application *App2* enters I/O phase, the data scheduler will check when the shared burst buffer is busy or not. If it finds that the shared burst buffer is occupied by *App1* and the PFS is idle, so it let *App2* to put its data directly to the PFS without buffering. When *App1* finishes buffer and go on to next compute phase, the burst buffer node will then flush the cached data to PFS. In that time, the shared burst buffer is idle and the scheduler let *App1* to redirect its I/O path to put its remaining data to shared burst buffer for higher available bandwidth. During the procedure, the metadata organizer monitor and provide the information of the usage of shared burst buffer and PFS all the time and reduce the possibility of multiple applications simultaneously

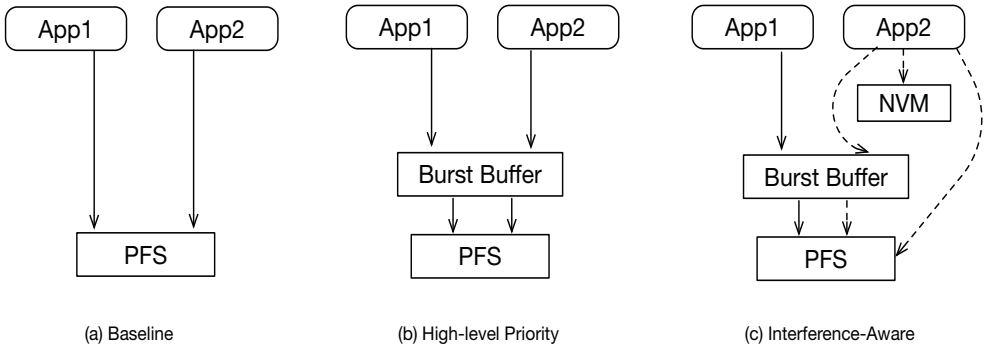

Fig. 4. Comparison of different I/O scheduling strategies.

occupying the same storage resource. If multiple applications concurrently access and share the underlying storage system, making both shared burst buffer and PFS busy in the same time. In this situation, data scheduler makes decision that data first buffer in compute node's local NVM and it will find an appropriate time to flush the cached data into next destination depending on the usage situation of shared burst buffer or PFS in the hierarchical storage system.

Algorithm 1: Interference-Aware Scheduling Algorithm

Input: $APP_1, APP_2, APP_3 ... APP_N$

```
1  ConcurrentAccessOnBB = 0;
2  ConcurrentAccessOnPFS = 0;
3  for APP_1 to APP_N do
4      Runtime captures I/O request from APP_i;
5      if ConcurrentAccessOnBB < N/2 then
6          ConcurrentAccessOnBB+=1;
7          Transfer data to shared burst buffers;
8          ConcurrentAccessOnBB-=1;
9      else if ConcurrentAccessOnPFS < N/2 then
10         ConcurrentAccessOnPFS+=1;
11         Transfer data to PFS;
12         ConcurrentAccessOnPFS-=1;
13     else
14         Store data in local NVM;
15     end
16 end
```

As shown in the Algorithm 1, we use the number of concurrent active application to estimate whether the the shared storage layers (i.e. shared burst buffer and PFS) are busy or not. For simplicity, if there are *N* applications are simultaneously running on the system, the data scheduler will allow a maximum of *N/2* application concurrently accessing to the same shared storage layers. For example, when APP_i issues an I/O request, the runtime intercepts this request and the data scheduler checks the load status of shared burst buffer. If the concurrent access number is smaller than *N/2*, data scheduler will set the destination of this I/O request to shared burst buffer (line 5-8). If the concurrent access number is larger than *N/2* which means the current load is high and the data scheduler will directly set the destination of this I/O request to PFS to reduce the burden of shared burst buffers (line 9-12). If both shared burst buffers and PFS are over-utilized in the same time, the data will be temporarily stored in the local NVM to be wait for next scheduling decision.

IV. Evaluation

We have carried out extensive experiments to validate the IAS design and verify the benefits of the proposed interference-ware scheduling strategy for managing concurrent I/O accesses to hierarchical HPC storage system. This section describes the experimental setup and discusses the experimental results.

A. Experimental Setup

The experiments were conducted on a cluster consisting of 16 nodes, in which 8 nodes were used as compute nodes, 4 nodes were used as SSD-based burst buffer nodes and 4 nodes were used as HDD-based parallel file system. Each compute node is equipped with two Intel Xeon E5-2630 CPU processors, 128GB RAM. Each burst buffer node is equipped with two Intel Xeon E5-2660 CPU processors, a 480GB SSD. Each PFS node is equipped with two Intel Xeon E5-2630 CPU processors, a 1.2TB hard drive. Each node runs Ubuntu 18.10 with the Linux kernel version 4.18. Compute nodes are connected via a Gigabit Ethernet. MPICH-3.3 release is installed on compute nodes. Burst buffer nodes and PFS nodes are installed with OrangeFS-2.9.7.

B. Synthetic Benchmark

We evaluate different I/O scheduling strategies on hierarchical storage system using our own synthetic benchmark that emulates common scientific application workloads such as alternation between computation and I/O phases. It uses POSIX-IO to issue write requests to the file system.

1) Performance with different ratio of I/O: In this experiment, we use two applications (each consist 32 processes and writes 1GB per process for each I/O phase). We repeat this pattern 5 times and use the average result. We vary the ratio of I/O over total run time from 0.8 to 0.2. We measure the average I/O time of the two applications in seconds. As we can be seen in Figure 5, with increasing ration of computation, the average I/O time of high-priority and Interference-Aware strategies decreased and interference-aware achieved the best performance. It is because with less I/O ratio, the more buffering time can be overlapped with more computing time. In summary, performance of interference-aware strategy performance is 4x and 2x higher than the baseline and the high-priority strategy we tested, respectively.

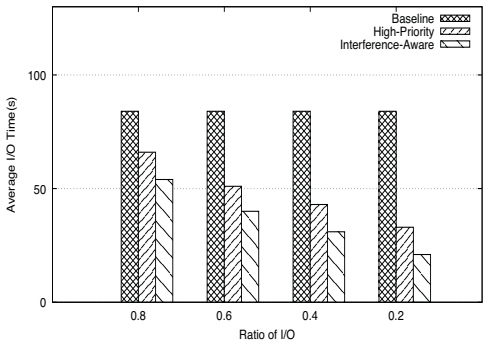

Fig. 5. Performance in terms of the ratio of I/O traffic of an application to the total running time.

2) Performance with different number of steps: In this experiment, we use two applications (each consist 32 processes and then writes 1GB per process for each I/O phase). We vary the number of steps for computation-I/O phases from 4 to 32. Figure 6 shows the results. We can see that with increasing steps, the average running time of high-priority and interference-aware strategies continue to decrease. Interference-aware strategy provides considerable performance improvement over the baseline and high-priority. It is because with more steps, the more performance benefit can get from the overlapping between data transfer time can and computing time. In general, in this test interference-aware strategy offers 4x and 2x higher performance when compared to no-buffer baseline and high-priority strategy separately.

3) Performance with different number of applications: In this experiment, we use varying number of applications (each consist 32 processes running on one compute node and then writes 1GB per process for each I/O phase) from 2 to 8. As it is illustrated in Figure 7, we can see that with increasing number of applications, the interference-aware average strategy outperforms baseline and high-priority. It is because interference-aware strategy can prevent too many applications simultaneously writing to same storage layer and thus effectively reduce performance loss due to I/O interference from cross-application. In conclusion, in this test

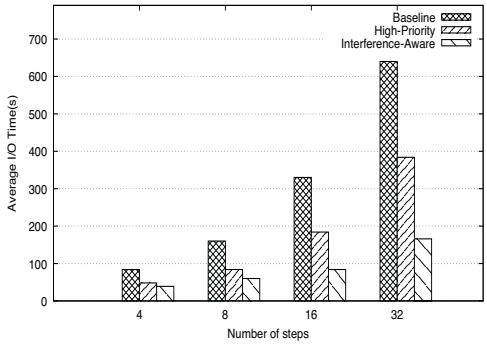

Fig. 6. Performance with varying steps.

interference-aware strategy offers 2.5x and 1.5x higher I/O performance when compared to no-buffer baseline and high-priority strategy respectively.

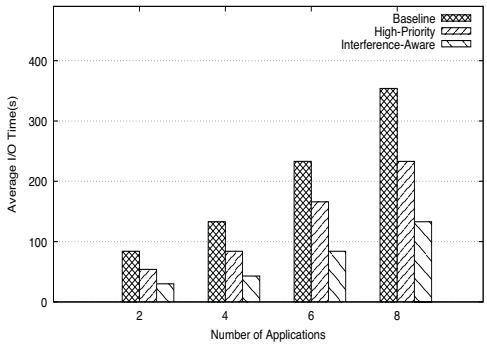

Fig. 7. Performance with varying number of applications.

C. Real-world Application

The Weather Research and Forecasting (WRF) model is a numerical weather prediction and atmospheric simulation system designed for both research and operational applications [14]. When a WRF application executes, it first reads the input data, then performs computations for weather simulation, periodically writes checkpoint data, and finally outputs the simulation results. It uses all processes to simultaneously perform the I/O operations which reflect the conventional feature of data-intensive application. In this experiment we concurrently run two instances of WRF and each uses 1, 2, 4 compute nodes respectively.

As can be seen from the results in Figure 8, as more compute nodes are used, the performances of interference-aware strategy are getting better. It is because interference-aware strategy can efficiently coordinate different application to reduce the chance of overload in certain one shared storage layer. In summary, these results demonstrate the advantage of the proposed interference-aware strategy .

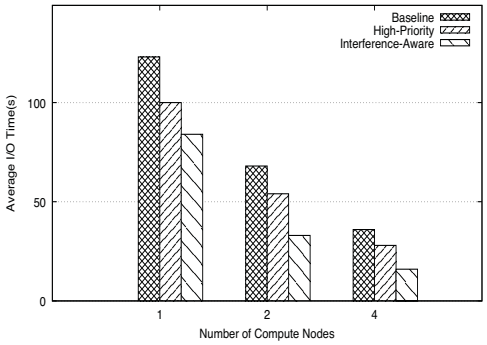

Fig. 8. Performance of WRF application.

V. Related Work

Numerous research studies have examined the burst buffer system and I/O Interference issue in HPC storage systems. In this section, we review these existing studies and compare them with this study.

A. Burst Buffer

Several studies [4], [15], [16] have demonstrated integrating burst buffer into HPC systems is a promising solution for addressing the I/O bottleneck for data-intensive applications. Chen et. al. [17] proposed an active burst buffer architecture to explore the computational capability by enhancing the existing burst buffer system with data analysis capabilities. Wang et. al. [18] studied node-local burst buffers to deliver scalable and efficient aggregation of I/O bandwidth for checkpoint and restart of data-intensive applications. Bent et. al. [19] also studied node-local burst buffers to deliver scalable write performance for local I/O requests. Kougkas et .al. [20] proposed a heterogeneous-aware I/O buffering system to enable transparent data movement for the multi-tiered storage system. Additionally, Han et. al. [21] proposed a user-level I/O isolation scheme to minimize the overhead for SSDs in burst buffers. Thapaliya et. al. [22] investigated I/O scheduling techniques as a mechanism to mitigate burst buffer I/O interference. In our work, we study the I/O scheduling problem for hierarchical HPC storage system including multiple storage layers.

B. I/O Interference

Numerous studies [23]–[25] focused on the I/O interference issue in HPC systems. Lebre et. al. [26] introduced a scheduling method to efficiently aggregate and reorder I/O requests. Zhou et. al. [27] proposed a novel I/O-aware batch scheduling framework to coordinate ongoing I/O requests on petascale computing systems. Gainaru et. al. [28] proposed several scheduling approaches to mitigate I/O congestion by analyzing the effects of interference on application I/O bandwidth. Herbein et. al. [29] proposed a batch job scheduling method to reduce contention by integrating I/O awareness into job scheduling policies. These efforts mainly focus on reducing I/O contention in parallel file system. In our study, we aim to address the I/O interference issue through the data scheduling in the hierarchical HPC storage system.

VI. Conclusion

In this paper, we propose IAS, a dynamic, interference-aware scheduling scheme that manages the data movement among different layers of hierarchical HPC storage system to coordinate concurrent data-intensive applications. By monitoring the status of applications' I/O activities and each layer of the storage system at the same time, IAS can make scheduling decisions for directions of data movement to reduce I/O interference from different applications and thus improve applications' I/O performance. By automatically placing data on multiple storage layers, such an efficient data management approach provide a promising, reliable solution for the I/O challenges towards the hierarchical storage architecture of future exascale HPC system. Extensive experiments on synthetic benchmark and real-world application have been conducted and the results have demonstrated that our proposed IAS strategy outperforms the existing strategies and improves application performance. In future work, we plan to strengthen this IAS approach, by utilizing the impact of different I/O patterns on different tiers of hierarchical storage system. The I/O access pattern will be important to characterize the applications and provide better performance improvements.

VII. Acknowledgment

We are thankful to the reviewers for evaluating this study and providing valuable feedback. This research is supported in part by the National Key Research and Development Program of China (2017YFB0202002), the National Science Foundation under grant CNS-1338078, CNS-1362134, CCF-1409946, and CCF-1718336.

References

[1] A. Choudhary, W.-k. Liao, K. Gao, A. Nisar, R. Ross, R. Thakur, and R. Latham, "Scalable I/O and Analytics," in *Journal of Physics: Conference Series*, vol. 180, no. 1. IOP Publishing, 2009, p. 012048.

[2] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig *et al.*, "The International Exascale Software Project Roadmap," *International Journal of High Performance Computing Applications*, vol. 25, no. 1, pp. 3–60, 2011.

[3] B. Peng, R. Nan, Y. Su, Y. Qiu, L. Zhu, and W. Zhu, "Five-Hundred-Meter Aperture spherical telescope project," in *International Astronomical Union Colloquium*, vol. 182. Cambridge University Press, 2001, pp. 219–224.

[4] 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.

[5] (2017) Cori Specifications. [Online]. Available: http://www.nersc.gov/users/computational-systems/cori/configuration

[6] (2017) Trinity Specifications. [Online]. Available: http://www.lanl.gov/projects/trinity/specifications.php

[7] (2018) Summit Specifications. [Online]. Available: https://www.olcf.ornl.gov/summit

[8] (2018) Sierra Specifications. [Online]. Available: https://hpc.llnl.gov/hardware/platforms/sierra

[9] (2018) Aurora Specifications. [Online]. Available: https://aurora.alcf.anl.gov/

[10] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu, "Six Degrees of Scientific Data: Reading Patterns for Extreme Scale Science IO," in *Proceedings of the 20th international symposium on High performance distributed computing*. ACM, 2011, pp. 49–60.

[11] S. S. Vazhkudai, B. R. de Supinski, A. S. Bland, A. Geist, J. Sexton, J. Kahle, C. J. Zimmer, S. Atchley, S. Oral, D. E. Maxwell *et al.*, "The Design, Deployment, and Evaluation of the CORAL Pre-Exascale Systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE Press, 2018, p. 52.

[12] (2017) Datawarp Manual. [Online]. Available: http://docs.cray.com/books/S-2558-5204/S-2558-5204.pdf

[13] W. Liang, Y. Chen, J. Liu, and 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.

[14] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, W. Wang, and J. G. Powers, "A description of the advanced research wrf version 2," National Center For Atmospheric Research Boulder Co Mesoscale and Microscale ..., Tech. Rep., 2005.

[15] K. Sato, K. Mohror, A. Moody, T. Gamblin, B. R. De Supinski, N. Maruyama, and S. Matsuoka, "A User-level Infiniband-based File System and Checkpoint Strategy for Burst Buffers," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014, pp. 21–30.

[16] J. Lofstead, I. Jimenez, C. Maltzahn, Q. Koziol, J. Bent, and E. Barton, "DAOS and Friends: A Proposal for an Exascale Storage System," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 50.

[17] C. Chen, M. Lang, L. Ionkov, and Y. Chen, "Active Burst-Buffer: In-Transit Processing Integrated into Hierarchical Storage," in *Networking, Architecture and Storage (NAS), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–10.

[18] T. Wang, K. Mohror, A. Moody, K. Sato, and W. Yu, "An Ephemeral Burst-Buffer File System for Scientific Applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 69.

[19] J. Bent, S. Faibish, J. Ahrens, G. Grider, J. Patchett, P. Tzelnic, and J. Woodring, "Jitter-Free Co-Processing on a Prototype Exascale Storage Stack," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE*

[27] Z. Zhou, X. Yang, D. Zhao, P. Rich, W. Tang, J. Wang, and Z. Lan, "I/O-Aware Batch Scheduling for Petascale Computing Systems," in *28th Symposium on*. IEEE, 2012, pp. 1–5.

[20] A. Kougkas, H. Devarajan, and X.-H. Sun, "Hermes: A Heterogeneous-Aware Multi-Tiered Distributed I/O Buffering System," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2018, pp. 219–230.

[21] J. Han, D. Koo, G. K. Lockwood, J. Lee, H. Eom, and S. Hwang, "Accelerating a Burst Buffer via User-Level I/O Isolation," in *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 2017, pp. 245–255.

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

[23] A. Uselton, M. Howison, N. J. Wright, D. Skinner, N. Keen, J. Shalf, K. L. Karavanic, and L. Oliker, "Parallel I/O Performance: From Events to Ensembles," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–11.

[24] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf, "Managing Variability in the IO Performance of Petascale Storage Systems," in *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*. IEEE, 2010, pp. 1–12.

[25] X. Zhang, K. Davis, and S. Jiang, "IOrchestrator: Improving the Performance of Multi-node I/O Systems via Inter-Server Coordination," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–11.

[26] A. Lebre, G. Huard, Y. Denneulin, and P. Sowa, "I/O Scheduling Service for Multi-Application Clusters," in *Cluster Computing, 2006 IEEE International Conference on*. IEEE, 2006, pp. 1–10.
Cluster Computing (CLUSTER), 2015 IEEE International Conference on. IEEE, 2015, pp. 254–263.

[28] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir, "Scheduling the I/O of HPC Applications under Congestion," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 1013–1022.

[29] S. Herbein, D. H. Ahn, D. Lipari, T. R. Scogland, M. Stearman, M. Grondona, J. Garlick, B. Springmeyer, and M. Taufer, "Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2016, pp. 69–80.