Contention-Aware Resource Scheduling for Burst Buffer Systems

Weihao Liang
University of Science and Technology of China
Hefei, China
lwh1990@mail.ustc.edu.cn

Yong Chen
Texas Tech University
Lubbock, TX, USA
yong.chen@ttu.edu

Jialin Liu
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
jalnliu@lbl.gov

Hong An
University of Science and Technology of China
Hefei, China
han@ustc.edu.cn

ABSTRACT

Many scientific applications in critical areas are becoming more and more data-intensive. As the data volume continues to grow, the data movement between storage and compute nodes has turned into a crucial performance bottleneck for many data-intensive applications. Burst buffer provides a promising solution for these applications by absorbing bursty I/O traffic to let applications return to computing phase quickly. However, the resource allocation policy for burst buffer is understudied, and the existing strategies may cause severe I/O contention when a large number of I/O-intensive jobs access the burst buffer system concurrently. In this study, based on the theoretic analysis of I/O model, we present a contention-aware resource scheduling (CARS) strategy that manages the burst buffer resource to coordinate concurrent jobs. Extensive experiments have been conducted and the results have demonstrated that the proposed CARS design outperforms the existing allocation strategies and improves both job performance and system utilization.

CCS CONCEPTS

• **Information systems → Hierarchical storage management**; *Record and buffer management*; • **Hardware** → *External storage*;

KEYWORDS

Burst Buffer, Runtime System, Resource Management, Scheduling Algorithm, I/O System, Modeling, I/O Contention

ACM Reference Format:
Weihao Liang, Yong Chen, Jialin Liu, and Hong An. 2018. Contention-Aware Resource Scheduling for Burst Buffer Systems. In *ICPP '18 Comp: 47th International Conference on Parallel Processing Companion, August 13–16, 2018, Eugene, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3229710.3229718

1 INTRODUCTION

More and more scientific applications in critical areas, such as climate science, astrophysics, combustion science, computational biology, and high-energy physics, tend to be highly data intensive and present significant data challenges to the research and development community [8, 9]. Large amounts of data are generated from scientific experiments, observations, and simulations. The size of these datasets can range from hundreds of gigabytes to hundreds of petabytes or even beyond.

Meantime, with the increasing performance gap between the computing and I/O capability, data access has become the performance bottleneck of many data-intensive applications, which usually contain a large number of I/O accesses and large amounts of data are written to and retrieved from storage system. The newly emerged burst buffer [14] is a promising solution to address this bottleneck issue. Given that the I/O patterns of many scientific applications are not regular but are bursty [16], burst buffer is designed as a layer of hierarchical storage, which comprises of high speed storage medium such as Solid State Drives (SSD), to let the applications quickly write or read data and return to the following computation phase as soon so possible. The data temporarily stored in burst buffer can then be transferred to the remote storage system asynchronously without interrupting the applications. In recent years, several peta-scale HPC systems have been deployed with burst buffer sub-system, such as Cori supercomputer [1] at National Energy Research Scientific Computing Center (NERSC) and Trinity [4] at Los Alamos National Laboratory (LANL). A number of large-scale data-intensive scientific applications have gained significant performance improvement from these burst buffer systems [6].

The burst buffer systems are designed as shared resources for multiple users and applications. Previous studies mainly focus on how to leverage burst buffer to improve the applications performance by reducing the I/O time [19, 23, 24] and overlapping between the computation and I/O phases of applications [7, 20]. However, the resource management for burst buffer is still understudied and the existing scheduling strategies only consider the capacity request of users, which may cause I/O contention between multiple concurrent I/O intensive applications.

In this paper, we present a preliminary study of a contention-aware resource scheduling (CARS) strategy to reduce the I/O contention for multiple concurrent I/O intensive jobs and improve the performance of burst buffer system. Following the study of different resource scheduling policies for shared burst buffer system and I/O modeling, CARS is designed as a new resource allocation and scheduling strategy for burst buffer. It characterizes the I/O activities of running applications on burst buffer system, and allocates the burst buffer resource for new jobs based on the analysis of the I/O load distribution among the burst buffer system to prevent or reduce interference from other jobs. Compared to the existing allocation strategies that only consider the capacity factor, the current experimental results show that CARS can significantly mitigate I/O contention between data-intensive jobs, speeding up the I/O process and improving the burst buffer system utilization.

The rest of paper is organized as follows. Section 2 discusses background and motivation of this research. Section 3 presents the design of the contention-aware resource scheduling and I/O modeling. Experimental and analytical results are presented in Section 4. Section 5 discusses the existing work and compares this study with them, and Section 6 concludes this study and discusses possible future work.

2 BACKGROUND AND MOTIVATION

In this section, we first introduce the background about the burst buffer architecture and existing resource allocation policies of shared burst buffer systems. Then we illustrate the motivation of this research study.

2.1 Overview of Burst Buffer

Figure 1 illustrates an architecture overview of a shared burst buffer system. The burst buffer resides on dedicated nodes that are usually deployed with SSDs as a high-speed storage tier. As a shared resource, the burst buffer nodes are available for all compute nodes to access directly. While applications from compute nodes issue bursty I/O operations, the burst buffer nodes can quickly absorb I/O requests in their local SSDs, and let the applications continue to the next computing phase as soon as possible. Afterward, the burst buffer nodes will transfer the data stored in their local SSDs to a parallel file system asynchronously. In this paper, we focus on the allocation policies and data transfer between compute nodes and burst buffer nodes. Existing large-scale HPC systems such as Cori [1] and Trinity [4] have deployed such shared burst buffer system architecture as production systems.

2.2 Resource Management for Burst Buffer

Burst buffer serves the entire compute system as a shared storage layer. Thus, it is crucial for resource scheduler to manage the burst buffer nodes efficiently to be accessed by different users or applications. In the burst buffer system of Cori or Trinity supercomputer, the DataWarp [2] software integrated with SLURM [3] workload manager is responsible for the resource management. When a user submits jobs by indicating the options for burst buffer requirement in the SLURM batch script, the resource scheduler will allocate burst buffer nodes to meet users' capacity need. The capacity request from users will be split up into multiple chunks and one chunk size is the minimum allocation size on one burst buffer node. For example, there are two choices of the chunk size on the Cori system, 20GB and 80GB.

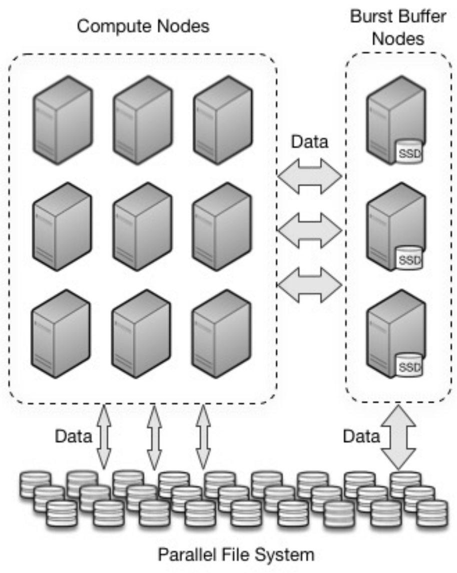


Figure 1: Shared burst buffer architecture.

There are mainly two allocation policies for how to assign burst buffer nodes for jobs in Cori, and users can manually specify the *optimization_strategy* option in SLURM batch script to chose different policies [2]. The first one is *bandwidth* policy that the resource scheduler will select as many burst buffer nodes as possible in round-robin fashion to maximize bandwidth for a job. In this case, users' data will be striped across several burst buffer nodes and one node contains one chunk of capacity request. So each node can be shared by several jobs under *bandwidth* policy. The second one is *interference* policy, in which the system will assign as few burst buffer nodes as possible to minimize interference from other jobs. In this case, a burst buffer node will comprise as many chunks as possible and be occupied by only one job at a time.

2.3 Motivation

While the existing *bandwidth* and *interference* scheduling policies serve the purpose to some extent, they are rudimentary and underdeveloped to well leverage the valuable burst buffer resources. Particularly, the *interference* policy is designed to minimize the contention from multiple jobs, but scarifies the utilization of burst buffer nodes due to an "exclusive" allocation policy. On the other hand, the *bandwidth* policy is designed to better utilize the burst buffer resources but do not consider the contention from highly concurrent jobs, which is the norm on large-scale HPC systems, especially for projected next-generation, exascale systems.

It is easy to understand the limitation of the *interference* policy, as such policy prevents resource sharing among jobs on any one burst buffer node. Thus the *interference* policy will be limited by the scale of the burst buffer system, i.e. the number of burst buffer nodes available to jobs, instead of the total capacity available, or the total bandwidth available. Such scheduling policy can easily lead to under-utilization of resources, including the under-utilization of both bandwidth and capacity.

The *bandwidth* policy, however, takes the other extreme. It only focuses on resource sharing but ignores the critical contention issue. As a comparison, the *interference* policy can lead to a scenario that, even though there is no any contention from other jobs, one burst buffer node is only allocated to one job exclusively, largely restricting the utilization of burst buffer and more importantly, only delivering limited bandwidth to each job. Such an observation inspire our research motivation in this study that the burst buffer resource scheduling strategies should well consider the contention issue, while maximizing the benefits of sharing resources among jobs to best leverage the bandwidth and capacity available from burst buffer nodes.

From the previous discussion, we have seen the potential of a new burst buffer resource scheduling strategy, which should meet two conflicting goals: maximizing resource utilization while minimizing the contention.

3 METHODOLOGY

This section describes a preliminary study of a contention-aware resource scheduling solution for burst buffer systems including an overview, the scheduling algorithm, and initial modeling and analysis.

3.1 Overview

In order to manage burst buffer resource efficiently and to coordinate a large number of jobs from different users, we introduce a contention-aware resource scheduling (CARS) strategy for burst buffer systems to promote application and system performance.

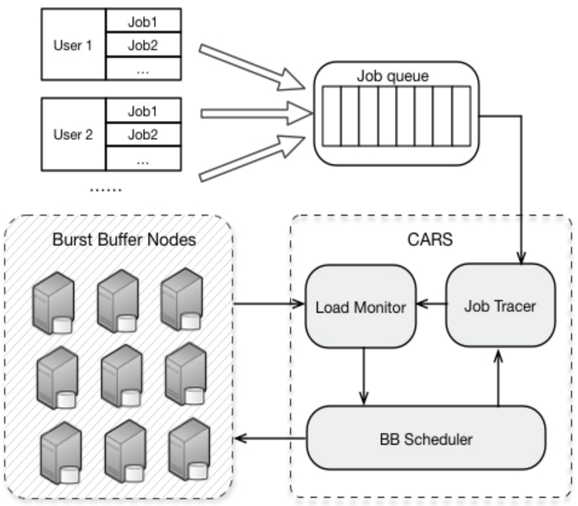


Figure 2: Overview of CARS Strategy.

Figure 2 shows a high-level view of the overall resource scheduling workflow and its integration with a typical HPC scheduling component. Users submit their jobs to the job queue and then the job scheduler (e.g., Slurm) will arrange the execution order of the jobs in the job queue. When there are jobs issued from the queue and accessing the burst buffer system, the *Job Tracer* will track how the current active jobs are distributed in the burst buffer system, such as on which burst buffer nodes each job is running. For instance, suppose a job A comprising of p_A processes has been allocated to BB1 and BB2, the *Job Tracer* collects the job information and adds a record like $A(BB1, BB2)$ into its database. Meantime, based on the status of active jobs monitored by *Job Tracer*, the *Load Monitor* will dynamically calculate and record the I/O load (e.g., number of concurrent I/O processes) of each burst buffer node. For example, by querying the database from *Job Tracer*, the *Load Monitor* will set the I/O load of BB1 and BB2 to $\frac{p_A}{2}$. When a new job from the job queue is ready to execute, the *BB Scheduler* gets the information from *Load Monitor* and determines which burst buffer nodes will be allocated to the new job by using a contention-aware scheduling algorithm, which will be described in detail in the next subsection. After the new job has been allocated burst buffer nodes, the *Job Tracer* and *Load Monitor* will update the corresponding state of current active jobs and burst buffer nodes.

3.2 Contention-Aware Resource Scheduling Strategy

To maximize resource sharing and utilization of burst buffer systems and minimize the I/O contention issue, we present a contention-aware scheduling strategy to allocate burst buffer nodes for concurrent jobs. The scheduling strategy is shown in Algorithm 1. The fundamental idea is to choose the most under-utilized burst buffer nodes for a new job to minimize the I/O contention from other active jobs. We use the number of concurrent active I/O processes to assess the I/O load of each burst buffer node. For example, suppose a job A is assigned to m burst buffer nodes and the number of I/O processes of job A is p_A. Among these m nodes, every $\frac{p_A}{m}$ processes of job A access one burst buffer node. When there are multiple jobs running on different burst buffer nodes, the resource scheduler can calculate the number of concurrent active I/O processes A_i for the ith burst buffer node. If a new job begins to execute, the scheduler will select the burst buffer node with minimum A_i (line 3) for this new job. If there are multiple burst buffer nodes with $minactive$ concurrent I/O processes, the algorithm will choose the one with the minimum index (line 5) of burst buffer node. Then the algorithm will update the A_i of the allocated burst buffer node (line 11). If the job needs n burst buffer nodes, it needs to run n iterations to complete the allocation for this new job.

3.3 Modeling and Analysis

In this section, we characterize and further study the proposed contention-aware resource scheduling (CARS) strategy through modeling and analysis. We also define evaluation metrics to understand the performance implications. We report detailed evaluation results in the next section. We first introduce parameters to characterize users' jobs and burst buffer systems, as shown in Table 1.

3.3.1 Modeling and Analyzing Burst Buffer Allocations. Given the *bandwidth* strategy and the newly proposed CARS strategy (described in Section 3.2), a job's request capacity will be striped across multiple burst buffer nodes, and each node contains at least one unit of minimum allocation capacity (i.e., G). Thus, for a particular capacity request (i.e., r_i) of the ith job, it will be allocated

Algorithm 1: Contention-Aware Resource Scheduling Algorithm

```
Input: $A_1, A_2, A_3 ... A_N$
Output: $allocatedlist$
1  Set $allocatedlist \leftarrow \emptyset$ ;
2  for 1 to $n$ do
3      $minactive = \min A_i$
4      if multiple $A_i$ equal to $minactive$ then
5          $newbb \leftarrow \min i$;
6      end
7      else
8          $newbb \leftarrow i$;
9      end
10     Add $newbb$ to $allocatedlist$;
11     Update $A_{newbb}$;
12 end
```

Table 1: Parameters/variables used in modeling and analysis

Symbol	Description
BM	Maximum bandwidth of one BB node
CM	Maximum capacity of one BB node
N	Total number of BB nodes
G	Minimum allocation capacity for one BB node
p_i	Number of I/O processes of the ith Job
n	Total numer of jobs
r_i	Request capacity size of the ith job

with maximum r_i/G burst buffer nodes. In *bandwidth* allocation policy described in Section 2, these burst buffer nodes are selected in round-robin fashion. We assume each job's I/O processes are evenly distributed among all their allocated burst buffer nodes. This is a reasonable assumption and can be easily achieved via the burst buffer management software (e.g. DataWarp) too. Therefore, when the ith job accesses the burst buffer system, there are $p_i G/r_i$ processes concurrently running on each burst buffer node. For the CARS strategy, we further model I/O contention and study its behaviors below. For the existing *interference* strategy, the behavior is rather simple, as each job gets allocated exclusive $\lceil \frac{r_i}{CM} \rceil$ burst buffer nodes.

3.3.2 Modeling and Analyzing I/O Contention. To study the impact of I/O contention, we assess the bandwidth for each process when there are multiple jobs concurrently accessing the burst buffer system. For simplicity and without losing generality, we assume each process accesses the burst buffer and can transfer data across the network at maximum I/O rate bm gigabytes per second. Suppose there are N processes simultaneously accessing one burst buffer node, the aggregate bandwidth $N \times bm$ can be achieved, if it does not exceed the maximum bandwidth of one burst buffer node (i.e., BM). If $N \times bm$ exceeds one burst buffer node's peak bandwidth BM, we assume the I/O bandwidth of this burst buffer node is equally shared by all concurrent processes from all jobs accessing it. Thus, the I/O bandwidth for each process can be computed as below.

$$I = \begin{cases} bm & \text{if } N \times bm \leq BM \\ \frac{BM}{N} & \text{if } N \times bm > BM \end{cases} \quad (1)$$

3.3.3 Evaluation Metrics. To evaluate the job performance and the overall system utilization, we present two evaluation metrics below. These evaluation metrics are orthogonal to each other and represent the most important two dimensions we focus on for evaluating burst buffer allocation strategies.

Average job efficiency. We define the efficiency of a job as the ratio of the I/O time when the job runs exclusively on one or more burst buffer nodes (denoted as TE) to the I/O time under I/O contention (denoted as TC). Suppose there are n jobs concurrently accessing the burst buffer system, the average job efficiency is defined as the geometric mean of all individual job efficiency, which can be calculated as:

$$Eff = \sqrt[n]{\prod_{i=0}^{n-1} \frac{TE_i}{TC_i}} \quad (2)$$

Average system utilization. Given N burst buffer nodes in total in the system and BM maximum I/O bandwidth for one node, during the ith time step (t_i), the aggregate I/O bandwidth of all current active jobs is BW_i, which can be calculated by summing each active job's I/O bandwidth using Equation 1 and Equation 2. We define the average system utilization for the burst buffer system as:

$$Util = \frac{\sum_i BW_i \times t_i}{N \times BM \times \sum_i t_i} \quad (3)$$

In Equation 6, the numerator can be described as the amount of data actually transfered during the entire I/O time of all jobs, and the denominator can be interpreted as the theoretical maximum amount of data transfered by using all burst buffer nodes. This metric $Util$ is system-oriented and assesses the bandwidth utilization of the entire burst buffer system.

4 PRELIMINARY EVALUATION

This section describes our preliminary evaluation results. To study the effectiveness of the proposed contention-aware resource scheduling on large-scale HPC systems, we conducted simulation evaluation and present the simulation tests based on the validated model with large-scale I/O workloads to further understand the performance implications.

4.1 I/O workload and BB system configuration

We conducted simulation experiments with the I/O workloads that represent large-scale applications. Based on Darshan I/O logs of petascale HPC system [18], we created a group of I/O workloads to mimic the real-world applications. Table 2 shows the workload configuration, including the number of I/O processes and the amount of data written.

In addition, we divided the workloads into two categories. The first category is consisted of large-scale jobs with more than 1,024

Table 2: Workload Configurations

Name	# of Procs	Write Data
Job1	8192	80TB
Job2	4096	40TB
Job3	2048	40TB
Job4	2048	20TB
Job5	1024	20TB
Job6	1024	10TB
Job7	512	8TB
Job8	256	4TB

processes. The second category is composed of medium- to small-scale jobs that use 1,024 processes or less. We further created different workload sets by combining different scales of workloads to represent the scenarios of a mix of jobs accessing the shared burst buffer system concurrently. In the simulation experiments, we used three sets of workload combinations described below.

- Set1: all jobs are large-scale jobs;
- Set2: all jobs are medium- to small-scale;
- Set3: a mixture of large-scale and medium- to small-scale jobs.

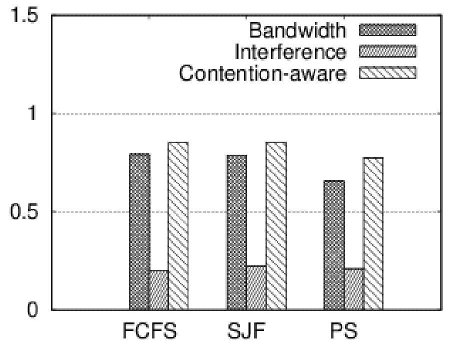
For the burst buffer system, we set the total number of burst buffer nodes as 256, the peak bandwidth per node as 6.5 GB/s, and the maximum capacity as 6.4 TB per node. This configuration is close to the configuration of the Cori burst buffer system [1].

4.2 Results and analysis

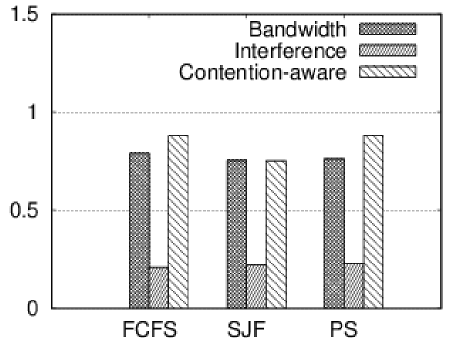
In the simulation experiments, two scales of jobs, 100 large-scale jobs or 1000 medium- to small-scale jobs, were used for each workload set. In addition, three job scheduling policies (FCFS, SJF, and PS) were simulated to decide the execution order of jobs in each workload set. For the FCFS (First come first serve) policy, we simply randomize the execution order of the jobs. For the SJF (Short job first) policy, jobs with less write data size were given higher priority for execution. For the PS (Priority scheduling) policy, jobs with larger writes were given higher priority to run. Each workload set for each job scheduling policy was repeated 5 times and we present the mean value in this section.

Figure 3 presents the average job efficiency of three workload sets separately. We first observe that the contention-aware strategy can improve the job efficiency compared to the other two policies in most cases. The exception happened in Set 2 using SFJ, which has almost the same job efficiency with the bandwidth policy. It is due to the similar allocation behaviors under these two different policies. We also observe that the contention-aware strategy achieved the largest improvement by more than 20% compared to the bandwidth policy in Set 3. The reason is that small-scale jobs can have sufficient I/O bandwidth they need and can finish quickly under the contention-aware policy, which in turn improved the average job efficiency.

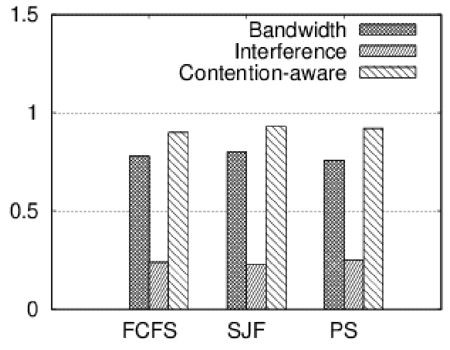
Figure 4 shows the average system utilization on three workload sets. This performance metric is system-oriented and we can clearly observe that our approach achieved higher system utilization on all workload sets, which indicates that the contention-aware policy is more efficient in utilizing the available bandwidth of burst buffer system than other policies. The highest system utilization is observed with SJF in workload set 3, where the system utilization achieved nearly 90%. This is most likely because most jobs can accurately select the underutilized burst buffer nodes to access and get enough bandwidth in this case.



(a) Set1

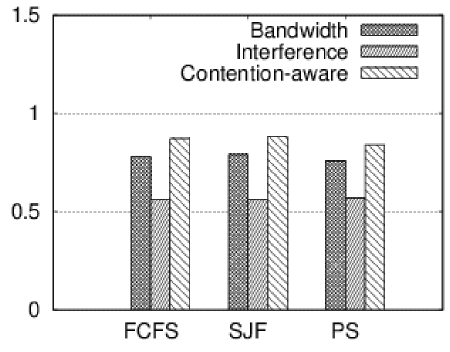


(b) Set2

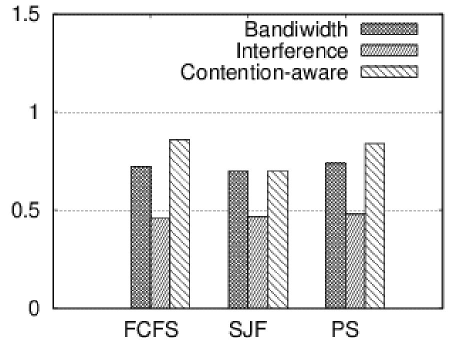


(c) Set3

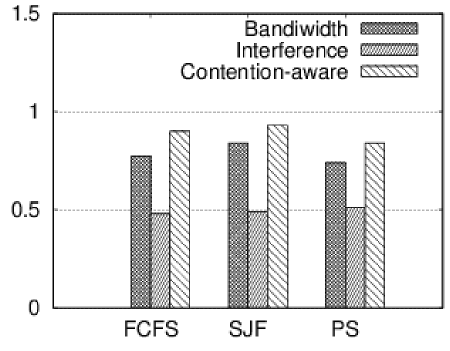
Figure 3: Average job efficiency



(a) Set1



(b) Set2



(c) Set3

Figure 4: Average system utilization

5 RELATED WORK

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

5.1 Burst Buffer

Several studies [14, 15, 19] 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. [7] proposed an active burst buffer architecture to enhance the existing burst buffer system with data analysis capabilities. Wang et. al. [23] 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. [5] also studied node-local burst buffers to deliver scalable write performance for local I/O requests. Our study focuses on the shared burst buffer systems that can be accessed by all compute nodes in the HPC system. Additionally, Han et. al. [11] proposed a user-level I/O isolation scheme to minimize the overhead for SSDs in burst buffers. Thapaliya et. al. [21] investigated I/O scheduling techniques as a mechanism to mitigate burst buffer I/O interference. In our work, we study the burst buffer resource allocation for the scenarios of multiple/many concurrent data-intensive applications.

5.2 I/O Contention

Numerous studies [17, 22, 25] focused on the I/O contention issue in HPC systems. Lebre et. al. [13] introduced a scheduling method to efficiently aggregate and reorder I/O requests. Zhou et. al. [26] proposed a novel I/O-aware batch scheduling framework to coordinate ongoing I/O requests on petascale computing systems. Gainaru et. al. [10] proposed several scheduling approaches to mitigate I/O congestion by analyzing the effects of interference on application I/O bandwidth. Herbein et. al. [12] 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 contention issue through the resource management in the burst buffer system.

6 CONCLUSION

In this paper, we have presented a resource scheduling strategy, called contention-aware resource scheduling (CARS), to mitigate the I/O contention and improve the performance of burst buffer systems. We have studied different resource scheduling policies in detail for the existing burst buffer systems and provided modeling and analysis about the I/O contention among concurrent jobs. Extensive experiments including both emulation and simulation evaluations have been conducted and the results have demonstrated that our proposed CARS strategy outperforms the existing allocation strategies and improves both job performance and system utilization. In future work, we plan to explore a more powerful I/O model based on the analysis of different I/O patterns in burst buffer systems.

7 ACKNOWLEDGMENT

We are thankful to the anonymous reviewers for their 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. W. Liang is also thankful to the scholarship support from the China Scholarships Council (CSC201706340082).

REFERENCES

[1] 2017. Cori Specifications. http://www.nersc.gov/users/computational-systems/cori/configuration/

[2] 2017. DataWarp Manual. http://docs.cray.com/books/S-2558-5204/S-2558-5204.pdf

[3] 2017. Slurm Wordload Manager. http://docs.cray.com/books/S-2558-5204/S-2558-5204.pdf

[4] 2017. Trinity Specifications. http://www.lanl.gov/projects/trinity/specifications.php/

[5] John Bent, Sorin Faibish, Jim Ahrens, Gary Grider, John Patchett, Percy Tzelnic, and Jon Woodring. 2012. Jitter-free co-processing on a prototype exascale storage stack. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*. IEEE, 1–5.

[6] Wahid Bhimji, Debbie Bard, Melissa Romanus, David Paul, Andrey Ovsyannikov, Brian Friesen, Matt Bryson, Joaquin Correa, Glenn K Lockwood, Vakho Tsulaia, et al. 2016. *Accelerating science with the NERSC burst buffer early user program.* Technical Report. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

[7] Chao Chen, Michael Lang, Latchesar Ionkov, and Yong Chen. 2016. Active burst-buffer: In-transit processing integrated into hierarchical storage. In *Networking, Architecture and Storage (NAS), 2016 IEEE International Conference on*. IEEE, 1–10.

[8] Alok Choudhary, Wei-keng Liao, Kui Gao, Arifa Nisar, Robert Ross, Rajeev Thakur, and Robert Latham. 2009. Scalable I/O and analytics. In *Journal of Physics: Conference Series*, Vol. 180. IOP Publishing, 012048.

[9] Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, et al. 2011. The international exascale software project roadmap. *International Journal of High Performance Computing Applications* 25, 1 (2011), 3–60.

[10] Ana Gainaru, Guillaume Aupy, Anne Benoit, Franck Cappello, Yves Robert, and Marc Snir. 2015. Scheduling the I/O of HPC applications under congestion. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 1013–1022.

[11] Jaehyun Han, Donghun Koo, Glenn K Lockwood, Jaehwan Lee, Hyeonsang Eom, and Soonwook Hwang. 2017. Accelerating a burst buffer via user-level i/o isolation. In *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 245–255.

[12] Stephen Herbein, Dong H Ahn, Don Lipari, Thomas RW Scogland, Marc Stearman, Mark Grondona, Jim Garlick, Becky Springmeyer, and Michela Taufer. 2016. 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, 69–80.

[13] Adrien Lebre, Guillaume Huard, Yves Denneulin, and Przemyslaw Sowa. 2006. I/o scheduling service for multi-application clusters. In *Cluster Computing, 2006 IEEE International Conference on*. IEEE, 1–10.

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

[15] Jay Lofstead, Ivo Jimenez, Carlos Maltzahn, Quincey Koziol, John Bent, and Eric Barton. 2016. 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, 50.

[16] Jay Lofstead, Milo Polte, Garth Gibson, Scott Klasky, Karsten Schwan, Ron Oldfield, Matthew Wolf, and Qing Liu. 2011. 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, 49–60.

[17] Jay Lofstead, Fang Zheng, Qing Liu, Scott Klasky, Ron Oldfield, Todd Kordenbrock, Karsten Schwan, and Matthew Wolf. 2010. Managing variability in the IO performance of petascale storage systems. In *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*. IEEE, 1–12.

[18] Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, Mr Prabhat, Suren Byna, and Yushu Yao. 2015. A multiplatform study of I/O behavior on petascale supercomputers. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 33–44.

[19] Kento Sato, Kathryn Mohror, Adam Moody, Todd Gamblin, Bronis R De Supinski, Naoya Maruyama, and Satoshi Matsuoka. 2014. 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, 21–30.

[20] Xuanhua Shi, Ming Li, Wei Liu, Hai Jin, Chen Yu, and Yong Chen. 2017. SSDUP: a traffic-aware ssd burst buffer for HPC systems. In *Proceedings of the International Conference on Supercomputing*. ACM, 27.

[21] Sagar Thapaliya, Purushotham Bangalore, Jat Lofstead, Kathryn Mohror, and Adam Moody. 2016. Managing I/O interference in a shared burst buffer system. In *Parallel Processing (ICPP), 2016 45th International Conference on*. IEEE, 416–425.

[22] Andrew Uselton, Mark Howison, Nicholas J Wright, David Skinner, Noel Keen, John Shalf, Karen L Karavanic, and Leonid Oliker. 2010. Parallel I/O performance: From events to ensembles. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 1–11.

[23] Teng Wang, Kathryn Mohror, Adam Moody, Kento Sato, and Weikuan Yu. 2016. 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, 69.

[24] Teng Wang, Sarp Oral, Michael Pritchard, Bin Wang, and Weikuan Yu. 2015. Trio: Burst buffer based i/o orchestration. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. IEEE, 194–203.

[25] Xuechen Zhang, Kei Davis, and Song Jiang. 2010. 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, 1–11.

[26] Zhou Zhou, Xu Yang, Dongfang Zhao, Paul Rich, Wei Tang, Jia Wang, and Zhiling Lan. 2015. I/O-aware batch scheduling for petascale computing systems. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. IEEE, 254–263.