Parallel Computing 58 (2016) 107–116

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

I/O-aware bandwidth allocation for petascale computing systems

Zhou Zhou[a,*], Xu Yang[a], Dongfang Zhao[a], Paul Rich[b], Wei Tang[b], Jia Wang[a], Zhiling Lan[a]

[a] Illinois Institute of Technology, Chicago, IL 60616, USA
[b] Argonne National Laboratory, Argonne, IL 60439, USA

ARTICLE INFO

Article history:
Received 29 November 2015
Revised 9 May 2016
Accepted 15 May 2016
Available online 16 May 2016

Keywords:
Job scheduling
Resource management
I/O congestion
Resource allocation

ABSTRACT

In the Big Data era, the gap between the storage performance and an application's I/O requirement is increasing. I/O congestion caused by concurrent storage accesses from multiple applications is inevitable and severely harms the performance. Conventional approaches either focus on optimizing an application's access pattern individually or handle I/O requests on a low-level storage layer without any knowledge from the upper-level applications. In this paper, we present a novel I/O-aware bandwidth allocation framework to coordinate ongoing I/O requests on petascale computing systems. The motivation behind this innovation is that the resource management system has a holistic view of both the system state and jobs' activities and can dynamically control the jobs' status or allocate resource on the fly during their execution. We treat a job's I/O requests as periodical subjobs within its lifecycle and transform the I/O congestion issue into a classical scheduling problem. Based on this model, we propose a bandwidth management mechanism as an extension to the existing scheduling system. We design several bandwidth allocation policies with different optimization objectives either on user-oriented metrics or system performance. We conduct extensive trace-based simulations using real job traces and I/O traces from a production IBM Blue Gene/Q system at Argonne National Laboratory. Experimental results demonstrate that our new design can improve job performance by more than 30%, as well as increasing system performance.

1. Introduction

We have already entered the age of petascale computing, and the insatiable demand for more computing power continues to drive the deployment of ever-growing high-performance computing (HPC) systems [1]. Today's production systems already comprise hundreds of thousands processors [2,3] and are predicted to have millions with exascale computing by 2018 [4,5]. Along with the rapid evolution of micro-processors, the explosion of the amount of data must be considered. In this so-called Big Data era, the data-sets generated by scientific applications are increasing exponentially in both volume and complexity [6–9].

* Corresponding author.
E-mail address: zzhou1@hawk.iit.edu (Z. Zhou).

http://dx.doi.org/10.1016/j.parco.2016.05.005
0167-8191/

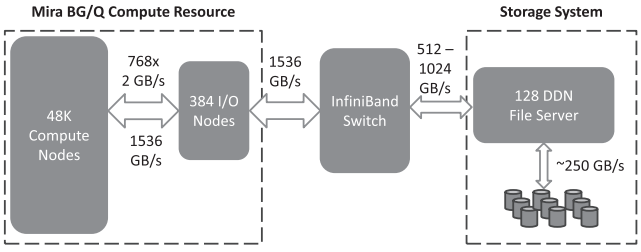


Fig. 1. Storage system architecture of the 10-petaflops Mira at Argonne national laboratory.

While the computing systems can leverage horizontal scale at an exponential rate in order to improve computing performance, the storage infrastructure performance is improving at a significantly lower rate. For example, the IBM Blue Gene/Q (BG/Q) system *Mira* at the Argonne Leadership Facility (ALCF) has a peak performance of 10 petaflops, which is 20 times faster than that of its predecessor *Intrepid* (0.5 petaflops), an IBM Blue Gene/P system [2,10,11]. But Mira's I/O throughput increases only 3 times compared with that of Intrepid [11]. As shown in Fig. 1, Mira's computing nodes can drive the I/O network at a full speed of 1536 GB/s whereas its storage system can deliver only 250 GB/s. This situation implies that even only one-quarter of the computing system can saturate all the bandwidth of its storage system and incur I/O congestion, potentially degrading application performance. Consequently, the increasingly large gap between an application's I/O requirement and the storage performance makes big data processing one of the most challenging problems faced by the computer science community.

To bridge this gap, researchers have conducted numerous studies from different perspectives. A well-known approach is to boost I/O performance through I/O middlewares (e.g., PLFS [12], DataStager [13], Damaris [14], IOrchestrator [15]) or I/O libraries (e.g., HDF5 [16], NetCDF [17]). As SSD becomes more cost-effective, memory-class storage is utilized to cache the temporary data from compute nodes in order to mitigate I/O bandwidth contention [18,19]. Additionally, researchers have proposed new architectural changes to push the I/O handling closer to the compute resource by adding a burst buffer tier on I/O nodes [20].

These studies typically focus on application-level optimization or are implemented on I/O nodes or file servers. From the application's perspective, some techniques are dedicated to individually optimizing this application's I/O access pattern. The I/O interference among multiple applications is not taken into account. On the other hand, from the system view, most approaches handle lower-level I/O requests in an uncoordinated manner without any knowledge from the upper-level applications [21]. For instance, when processing I/O requests, the storage server attempts to establish a fair share of throughput among multiple concurrent applications, which may lead to unexpected application performance degradation. Moreover, because of the lack of a global view of all ongoing I/O activities from different applications, important system-wide performance metrics—such as average job turnaround time and system utilization—are often overlooked.

In this work, we propose to address the I/O congestion problem through I/O-aware bandwidth allocation on I/O servers. Fig. 2 shows the overview of our solution. This solution extends the current HPC scheduling framework by integrating the runtime monitor of system states and I/O activities as well as runtime I/O control. The idea behind this is straightforward that once I/O congestion is detected, all concurrent I/O requests will be scheduled or coordinated like normal jobs on HPC systems. Based on this solution, we implement it in the I/O scheduler on the storage side as an extension to the current scheduling framework. The new framework not only selects queued user jobs for execution but also coordinates job I/O activities during their execution. Every time a new I/O is forwarded to the I/O server, the resource manager monitors all I/O requests being processed and checks whether the aggregate bandwidth requirement exceeds the maximum storage bandwidth capacity. If so, only part of the jobs are assigned with available bandwidth to conduct their I/O operations so as to minimize the impact caused by I/O congestion.

In [22] we have proposed an alternative solution implemented on the batch scheduler based on the open-source resource management software named Cobalt [23]. In the case of I/O congestion, the batch scheduler dynamically reacts using the runtime information to coordinate these concurrent I/O requests. The scheduling decision is made based on certain scheduling objectives. The basic idea is to dynamically coordinate concurrent jobs' I/O activities (e.g., running or suspending) to avoid I/O congestion.

In this paper we focus on the bandwidth allocation framework which is implemented on the I/O scheduler on storage infrastructure. We make the I/O scheduler on the I/O servers dynamically allocate bandwidth to jobs. The I/O scheduler monitors all the I/O requests forwarded to the I/O server and decides the bandwidth allocation plan on the fly. Similar to the I/O-aware scheduling method, jobs allocated with available bandwidth can continue their I/O operations. Other jobs without available bandwidth stay waiting in the I/O server. The bandwidth allocation plan is made based on certain objectives that are described in detail in the next subsection. The basic idea is to assign bandwidth to jobs in a certain order in which jobs are prioritized using a specific strategy.

More specifically, this paper makes the following contributions:

1. We present an I/O-aware bandwidth allocation framework for petascale computing that encompasses job abstraction, machine model, I/O congestion model and bandwidth allocation policies. In particular, several bandwidth al-

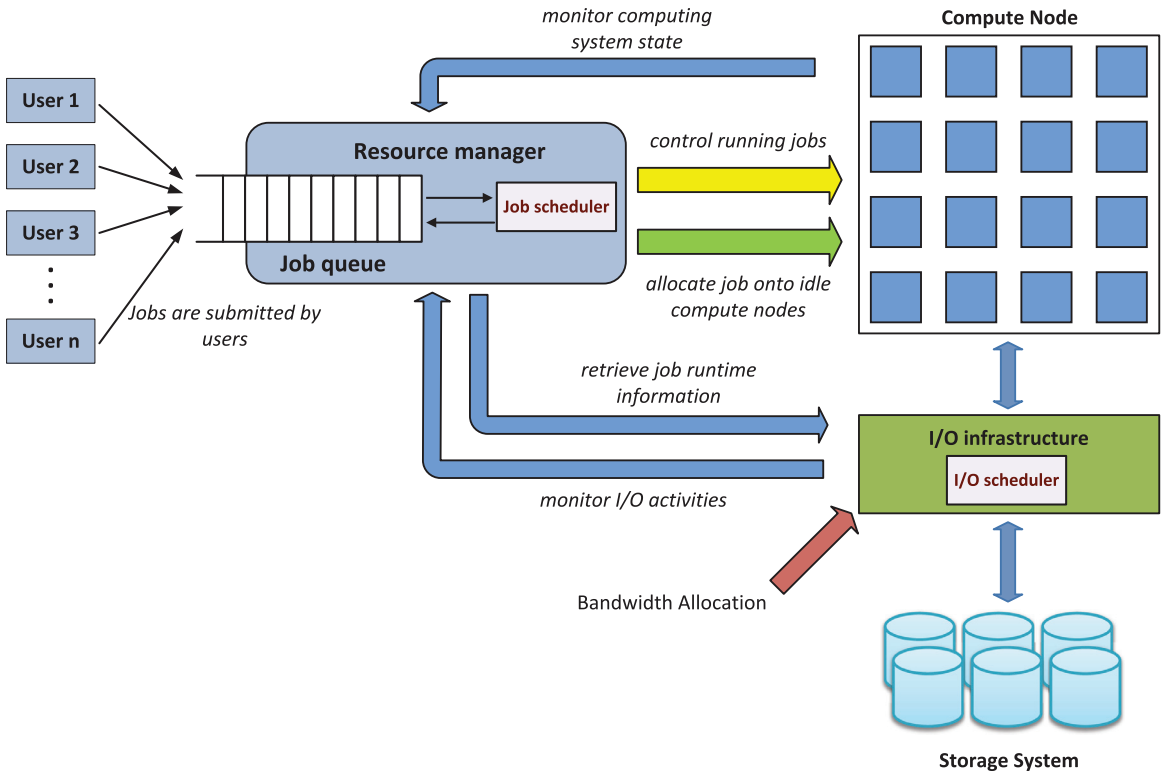


Fig. 2. Our I/O-aware scheduling framework with three new features: runtime I/O monitoring (blue arrow) dynamic control of running jobs (yellow arrow) and I/O-aware bandwidth allocation (red arrows).

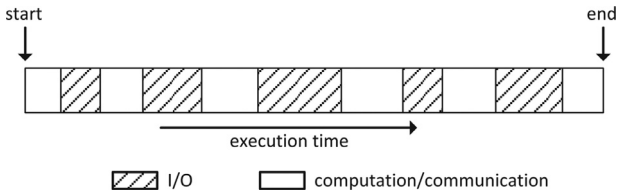


Fig. 3. Lifecycle of an HPC scientific application.

location policies are designed for the framework, each for achieving different objectives (e.g., fairness, lower job slowdown).

2. We conduct extensive trace-based simulations by using real job traces and I/O traces from the production 10-petaflops Mira system [24]. Experimental results demonstrate that our design improves job performance by more than 30% and also improves system performance.

The remainder of this paper is organized as follows. Section 2 describes our design of the I/O-aware bandwidth allocation framework. Section 3 presents experimental results of our trace-based simulation. Section 4 discusses related work. Section 5 concludes this paper and points out the future work.

2. Methodology

In this section, we present our I/O-aware bandwidth allocation framework implemented on the I/O scheduler.

2.1. Job abstraction

A typical HPC job often contains three types of activity: computation, communication, and I/O. Fig. 3 shows the lifecycle of a sample job. The job's runtime is split into phases for different activities, which repeatedly run during the job lifecycle. The horizontal axis represents the execution time. The height usually represents the job's resource requirement (e.g., nodes, memory, bandwidth). This abstraction provides a high-level view of the application behavior. For example, the I/O chunk in

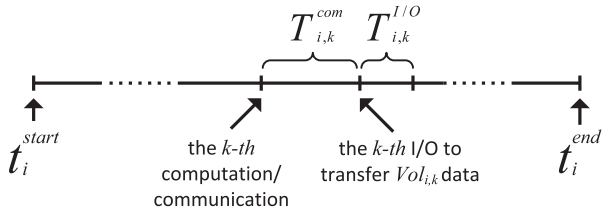


Fig. 4. Runtime phases of the ith job J_i requiring N_i nodes.

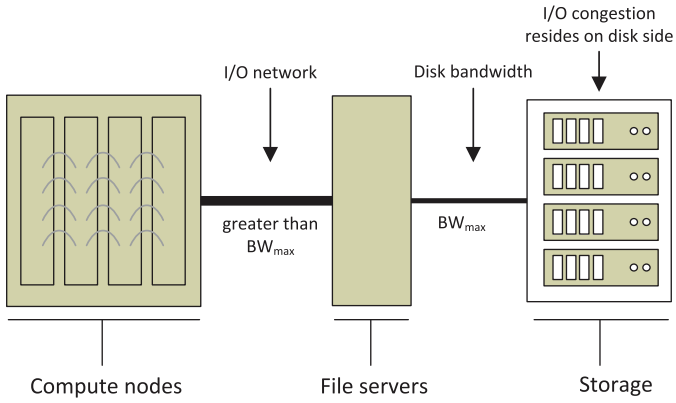


Fig. 5. Simplified I/O infrastructure model of petascale HPC systems.

Fig. 3 may contain several consecutive I/O calls (e.g., a loop of write accesses). In our job abstraction, this series of I/O calls is regarded as a single I/O request. Notice that our target platform is the IBM BG/Q system, which uses a partition-based resource allocation scheme [25]. The computation and network resource in a partition are dedicated to serve the job running on it. Hence, once a job is scheduled to run, the time consumed in computation and communication is fixed. As such, we unify computation and communication into one type of activity in Fig. 3. The time for I/O activity is substantially variable because of potential I/O congestion.

In Fig. 4 we depict the runtime phases of a job. Each running job J_i is associated with two basic attributes: *start time* as t_i^{start} and *size* (in nodes) as N_i. A job shows a periodical running pattern that computation/communication interleaves with I/O. We assume that each computation/communication is followed by an I/O request that transfers a chunk of data from or to the storage system via the I/O network. Therefore, each job J_i consists of n_i computation/communication and n_i I/O activities. The kth computation/communication activity of job J_i requires time $T_{i,k}^{com}$ to finish, and the kth I/O needs to transfer $Vol_{i,k}$ of data (in gigabytes).

2.2. System model

As shown in Fig. 5, we assume an HPC system composed of N compute nodes, each installed with the same number of identical uniform-speed processors. Each compute node is connected to the I/O nodes and can transfer data to the storage system across the I/O network at maximum bandwidth b (in gigabytes per second). The aggregate bandwidth of the whole system $b \times N$ is guaranteed to be less than the maximum capacity of the I/O network. Inside the storage system, we assume a group of file servers connected to a centralized parallel file system with a maximum bandwidth $BW_{\max}$. In many systems such as Intrepid and Mira, this maximum bandwidth $BW_{\max}$ is always less than the compute nodes' aggregate bandwidth $b \times N$ and is usually subject to the upper-bound maximum speed of the hard drives. More specifically, the rotation speed of the disk heads dominates the actual access speed of the storage system. Therefore, in this model we assume that the I/O congestion takes place at the storage side if the aggregated bandwidth of all active compute nodes $b \times N_{\text{active}}$ exceeds the total bandwidth capacity $BW_{\max}$ of the storage system.

2.3. Problem model

Fig. 6 shows the problem model regarding I/O congestion based on the aforementioned job abstraction and the system model. The x-axis represents the execution time, the y-axis the aggregated bandwidth. Suppose three jobs J_1, J_2, and J_3 are running concurrently on separated sets of nodes and each job has its own I/O pattern along with different bandwidth usage, illustrated as the height of each rectangle. Naturally a job's I/O operations may overlap with those of others. At this point, multiple jobs are accessing the storage system simultaneously, and inevitably they have to compete for I/O bandwidth with

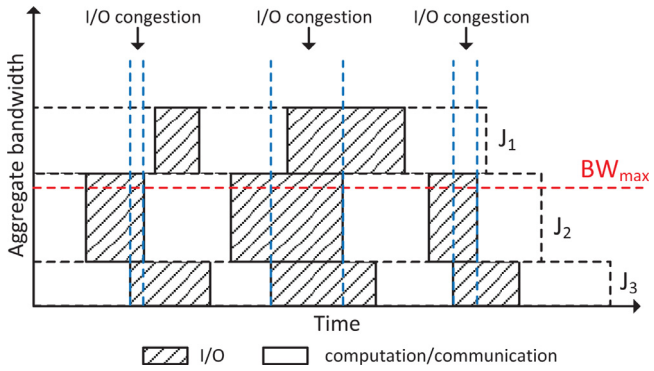


Fig. 6. Problem model.

one another. If the aggregate bandwidth of these three jobs exceeds the maximum bandwidth BW_{max} denoted by the red dashed line, I/O congestion occurs and impacts these jobs' I/O performance if no congestion control is taken.

This problem model is simple, but it reflects the motivation of this work: to utilize the resource management system to mitigate I/O congestion by monitoring and controlling the jobs' I/O operations on the fly. After pulling information about all the ongoing I/O operations, the I/O scheduler decides the bandwidth usage among jobs within a certain length of time. In other words, the I/O bandwidth is considered as a "schedulable" resource and shared by all the jobs.

2.4. I/O congestion model

In the system model of Fig. 5, we assume that the I/O network bandwidth is much larger than that of the storage system. Thus, the I/O congestion actually locates at the hard disk side, which is limited by the disk rotation speed. The lower-level I/O scheduler in the storage server may apply a simple, independent "first-in-first-out" policy on incoming I/O requests or a more elaborated strategy, such as minimizing disk-head movements by aiming at better data locality. From the system view, the storage system is accessed in a fair-sharing mode. We define a general I/O congestion model on HPC systems as follows: If $b \times N_{\text{active}} \geq B$, the actual bandwidth allocated to each compute node is $\frac{B}{N_{\text{active}}}$. Although extra overhead such as the time for disk-head movements should be considered, it is hard to precisely calculate the overhead in an analytical model. Therefore, in this work we set up a fair-sharing storage model with no extra overhead.

2.5. I/O-Aware Bandwidth Allocation Framework

To mitigate I/O congestion and improve system performance, we designed a novel I/O-aware bandwidth allocation framework to coordinate concurrent I/O requests. As shown in Fig. 2, the new scheduling framework extends the current HPC scheduling framework by integrating the runtime monitoring of system state and I/O activities and runtime I/O control. For the purpose of system efficiency, we implement the bandwidth allocation as an extension to the I/O scheduler on the I/O server. The key idea of our solution works as follows:

1. Every time a new I/O request is forwarded to the I/O server, the I/O scheduler monitors all I/O requests being processed and checks whether the aggregate bandwidth requirement exceeds the maximum value $BW_{\max}$.
2. If the storage bandwidth is overloaded, I/O congestion is likely to happen. The I/O scheduler pulls information of running jobs from the scheduling system and decides how to allocate I/O bandwidth to jobs.
3. The decision is applied on the fly during system running. Similar to batch job scheduling, the I/O scheduler dynamically allocate bandwidth among multiple jobs. Jobs assigned available bandwidth can perform their I/O operations. Jobs without bandwidth can be seen as temporarily "suspended" and should wait until next decision making iteration.

2.6. Job performance quantification

First we present two user-oriented metrics to quantify job performance regarding I/O congestion.

- **InstantSlowdown (InstSld)**: Suppose at time t, job J_i is performing the kth I/O operation. It has a total of $Vol_{i,k}$ data to transfer and has already transferred $W_{i,k}$ data. Let $t_{i,k}^{I/O}$ be the start time of the kth I/O operation. We define $InstSld_{(i,k)}(t)$, the instant slowdown of the kth job J_i at time t as

$$InstSld_{(i,k)}(t) = \frac{b \times N_i \times (t - t_{i,k}^{I/O})}{W_{i,k}}, \tag{1}$$

where $t - t_{i,k}^{I/O}$ is the elapsed time from the start time of current I/O operation to now and $b \times N_i \times (t - t_{i,k}^{I/O})$ is the theoretically maximum total size of data job J_i could transfer by now within this I/O operation, in other words, an ideal case without any I/O congestion. The ratio of this maximum size of data to the already transferred $W_{i,k}$ data represents the slowdown of transferring data caused by I/O congestion. Obviously, $InstSld_{(i,k)}(t) \geq 1$ with $InstSld_{(i,k)}(t) = 1$ indicating the data transferring is not interfered with by any I/O congestion.

- **AggregatedSlowdown (AggrSld)**: Again, we assume job J_i is performing its kth I/O operation. We define $AggrSld_{i,k}(t)$, the aggregate slowdown of job J_i at time t as

$$AggrSld_{(i,k)}(t) = \frac{t - t_i^{start}}{\sum_{j \leq k} T_{i,j}^{com} + \sum_{j<k} T_{i,j}^{I/O}}, \tag{2}$$

where $T_{i,j}^{com}$ is the time spent on the jth computation/communication operation and $T_{i,j}^{I/O}$ the time of the jth I/O operation without I/O congestion. Here $t - t_i^{start}$ equals the total elapsed time between the job start and the present time; and $j \leq k$ is the number of computation/communication or I/O the job has executed at time t since the start time t_i^{start}. The *AggrSld* reflects the extent of an application's delay due to I/O congestion.

2.7. Bandwidth allocation policies

Suppose K jobs in the system are performing I/O or are ready to perform I/O. Our scheduler selects a subset from the K jobs to perform I/O. Jobs that are not selected are suspended and wait until the next scheduling cycle. We propose three bandwidth allocation policies on the I/O scheduler. Every time the I/O scheduler chooses a job and allocates bandwidth as much as possible to satisfy its I/O requests. The maximal bandwidth a job can have equal to $b \times N$ which is the theoretical bandwidth it could drive over the I/O network. The detailed policies are as follows:

1. **FCFS**: The *FCFS* (first-come-first-serve) policy works the way a traditional job scheduler does. It allocates bandwidth to jobs in chronological order based on the start time of concurrent I/O requests. Jobs performing I/O with earlier start time have higher priority and are favored by the I/O scheduler.
2. **MinInstSld**: The *MinInstSld* policy favors jobs with low *InstSld* value. This method is close to *FCFS*, which orders jobs before making decision.
3. **MinAggrSld**: Similar to the *MinInstSld*, the *MinAggrSld* favor jobs with low *AggrSld* value.

3. Evaluation

3.1. Qsim Simulator

Qsim is an event-driven scheduling simulator for Cobalt [23,25]. Taking the historical job trace as input, Qsim quickly replays the job scheduling and resource allocation behavior and generates a new sequence of scheduling events as an output log. Qsim uses the same scheduling and resource allocation code that is used by Cobalt and thus has been proven to provide accurate resource management and scheduling simulation [25–27]. Qsim is open-source and available along with the Cobalt code releases [23].

3.2. Job Trace and I/O Trace

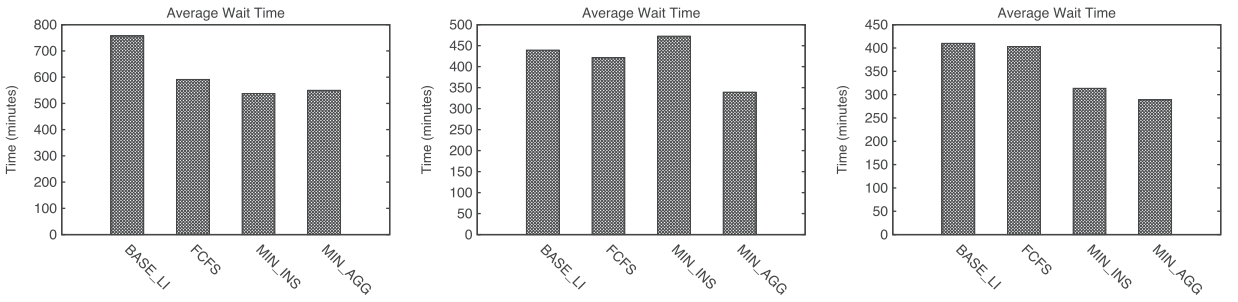
We evaluate our design by means of actual workload traces from the production BG/Q machine *Mira* at ALCF. We have collected a three-month job trace from 2014. The job trace provides rich information for our simulation, including submission time, job size, duration, and walltime.

To obtain information of I/O activities, we use the Darshan log collected from Mira [28]. Darshan is a user-level system tool to allow users to characterize the I/O behavior in an efficient and transparent manner at extreme scales. In essence, for each application running on the systems, Darshan collects the I/O footprint and summarizes it in a compact and statistically reproducible manner. Because it stores only the statistical summary of many I/O operations, the space and I/O overhead when enabling Darshan logs are minimum. By default, Darshan is turned on for all applications on the Blue Gene systems and is completely transparent to the users; no application change is in need. The effectiveness of Darshan logs has been demonstrated at extreme scales—up to 64K nodes.

By combining the job trace and I/O trace via pairing, we have built a comprehensive workload that contains job information (e.g., submission, runtime, size) as well as their I/O characteristics (e.g., number of I/O calls, I/O time, read and write transfer size). We divide the 3-month workload trace into three single-month workload traces and evaluate our design on each of them. This approach enables us to evaluate our design under workloads with different characteristics.

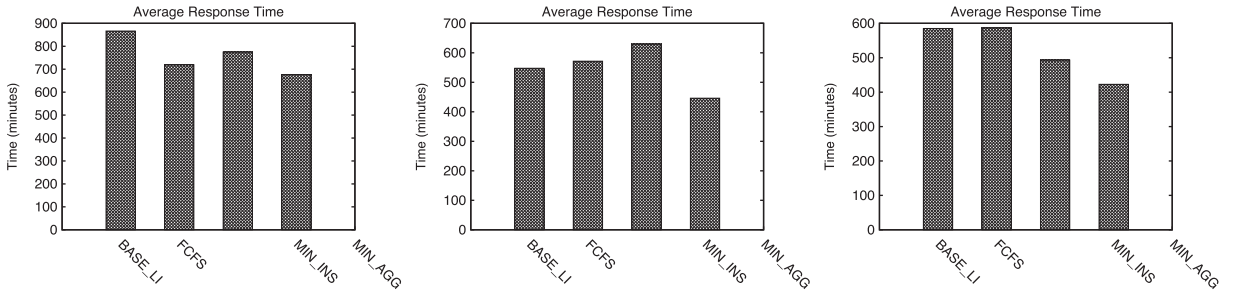
3.3. Evaluation metrics

Three widely used metrics are evaluated:



(a) Workload 1 (b) Workload 2 (c) Workload 3

Fig. 7. Average wait time with bandwidth allocation.



(a) Workload 1 (b) Workload 2 (c) Workload 3

Fig. 8. Average response time with bandwidth allocation.

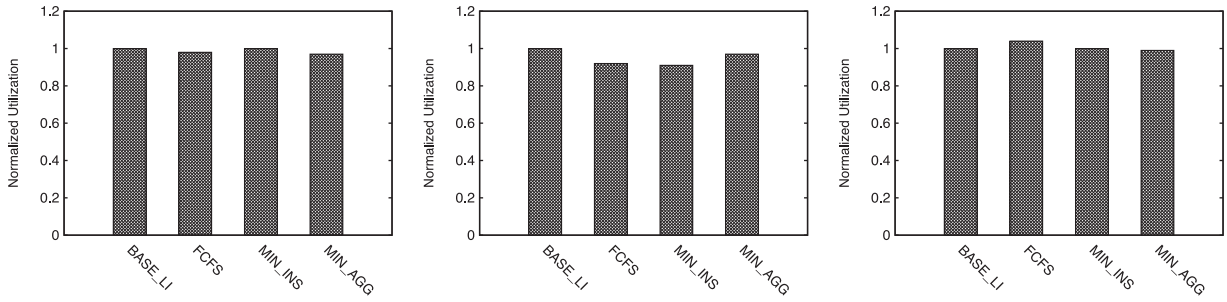
- *Average job wait time*. This metric denotes the average time elapsed between the moment a job is submitted and the moment it is allocated to run. It is commonly used to reflect the "efficiency" of a scheduling policy.
- *Average response time*. This metric denotes the average time elapsed between the moment a job is submitted and the moment it is completed. Similar to the above metric, it is often used to measure scheduling performance from the user's perspective.
- *System utilization*. System utilization rate is measured by the ratio of busy node-hours to the total node-hours during a given period of time [29] [30]. The utilization rate at the stabilized system status (excluding warm-up and cool-down phases of a workload) is an important metric of how well a system is utilized.

3.4. Results

3.4.1. Bandwidth Allocation Policies

Fig. 7 shows the average wait times of different bandwidth allocation policies used by the I/O scheduler. First it is clear that a bandwidth allocation policy aware of I/O-congestion is much more efficient than the *BASE_LINE* policy, which allocates bandwidth in a fair-share manner. Second, on all three workloads, the *MIN_AGGR_SLD* policy performs the best in reducing wait time than to the other policies. On both workload 2 and wordload 3, the *MIN_AGGR_SLD* policy reduce the wait time by more than 20%, while the *MIN_INST_SLD* policy even increases the wait time of workload 2. The *MIN_AGGR_SLD* focuses on the overall application performance rather than focusing on the local optimum of current I/O request like *MIN_INST_SLD* does. That is the reason that *MIN_AGGR_SLD* can achieve better reduction of wait time. Another observation is the *FCFS* policy has no stable advantage over the *BASE_LINE*. Only on workload 1, *FCFS* has improved the wait time by nearly 20%.

Fig. 8 shows results of average response time similar to Fig. 7. Both *MIN_INST_SLD* and *MIN_AGGR_SLD* have lower response times than do the other two policies. The largest reduction on reponse time is achieved on workload 3 by more than 30% with *MIN_AGGR_SLD*; and *MIN_INST_SLD* can reduce the response time by more than 20% on workload 1 and 3. We also observe that using *FCFS* cannot always improve the response time and may even have a negative impact on it. For example, on workload 2 and 3 we not that *FCFS* slightly increases the response time. In particular, *FCFS* are ineffective on lowering the response time, showing nearly the same value as the baseline result. According to the definition, the job response time consists of wait time and job runtime. While the decrement of wait time benefits from our I/O-aware bandwidth allocation policies, a job may not be assigned with sufficient bandwidth. This situation increases the running time, which is likely to diminish the benefit on wait time.



(a) Workload 1 (b) Workload 2 (c) Workload 3

Fig. 9. Normalized system utilization.

Fig. 9 shows the system utilization on three workloads. For convenience, we normalize the absolute utilization value based on the results of *BASE_LINE*. The largest utilization drop is seen with *FCFS* and *MIN_INST_SLD* on workoad 2, where the utilization decreases by nearly 10%. Other bandwidth allocation policies have negligible loss on utilization, which is treated as acceptable cost. Taking all the results into consideration with Figs. 7 and 8, the *MIN_AGGR_SLD* policy is the best trade-off and achieves a good balance between user satisfaction and system utilization.

4. Related work

One traditional approach to addressing the I/O bottleneck on extreme-scale systems is to employ high-level I/O libraries, such as HDF5 [16] and NetCDF [17]. These libraries stipulate the data format of the applications as well as their I/O interfaces. Other researchers proposed several loosely coupled middleware solutions, such as PLFS [12], DataStager [13], and Damaris [14]. These systems work independently of both the underlying storage systems and the upper-level applications and thus greatly generalize the applicability. Nevertheless, an obvious drawback of this approach is the potential overhead introduced by the middleware. More recently, the trend of solving the I/O imbalance in extreme-scale systems is to move the data closer to the compute resource. Ning et al. [20] propose moving file handling to the I/O nodes in order to ameliorate the I/O pressure from the massive number of compute nodes.

Performance variability of parallel applications in HPC systems has been existing for a long time and attracted a lot of research. Such variability is due to the fact that concurrently running parallel applications interfere with each other by sharing a great mount of system resources such as network and I/O channels. Bhatele et al. show that job runtime is influenced by network sharing, and the performance variability introduced by network contention could be as high as 70% [31].

I/O contention in HPC systems draws considerable attention in the community because it is the root cause of parallel applications' performance variability [32–34]. Zhang et al. and Lofstead et al. schedule each application's I/O request individually without a global view from system's perspective [33,34]. Their solutions require support from specific I/O management in the system level for better results. Solutions to the I/O contention between parallel applications have been studied in various works [35–37]. Dorier et al. analyze the I/O interference between two applications [21]. They make a quantified study of performance improvement obtained by interrupting or delaying either one in order to avoid I/O contention. Gainaru et al. analyze the effects of interference on application I/O bandwidth and propose several scheduling techniques that apply to system I/O level to mitigate congestion [38].

5. Conclusion

In this paper, we have presented an I/O-aware bandwidth allocation framework to alleviate the I/O congestion on petascale HPC systems. In our design, the I/O congestion scenario is formalized into a classical scheduling problem by treating I/O bandwidth as schedulable resource and having the system dynamically schedules these I/O accesses. The bandwidth allocation method moves the task to the I/O scheduler on the I/O server and releases the batch scheduler from controlling jobs on the fly. This is more effective because processing I/O requests on I/O servers causes much less overhead than suspending/restarting jobs via the resource management system. Our trace-based simulations clearly demonstrate the performance benefit obtained with these new policies.

While this study targets Mira, our design is generally applicable to other HPC systems. To the best of our knowledge, this is the first work on addressing I/O congestion from the perspective of resource management. Several avenues remain open for future work. One is to build a model to predict an application's I/O behavior based on its past I/O trace. Given the I/O pattern in advance, the job scheduler can take this into consideration and make decisions to achieve better performance. We also plan to investigate more complicated algorithm to obtain more optimal solutions. In addition, we plan to expand

this work by developing a smart resource management framework for better managing nontraditional resources including I/O and power consumption.

Acknowledgments

The work at Illinois Institute of Technology is supported in part by US National Science Foundation grants CNS-1320125 and CCF-1422009. This research used data of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

[1] Top500 supercomputing web site.
[2] Mira at ANL, http://www.alcf.anl.gov/mira, Accessed September 5, 2014.
[3] Titan at ORNL, https://www.olcf.ornl.gov/titan/, Accessed September 5, 2014.
[4] Exascale computing predicted by 2018, http://www.computerworld.com/article/2550451/computer-hardware/scientists-it-community-await-exascale-computers.html, Accessed September 5, 2014.
[5] K. Wang, K. Brandstatter, I. Raicu, Simmatrix: Simulator for many-task computing execution fabric at exascale, in: Proceedings of the High Performance Computing Symposium, Society for Computer Simulation International, 2013, p. 9.
[6] P.A. Freeman, D.L. Crawford, S. Kim, J.L. Munoz, Cyberinfrastructure for science and engineering: Promises and challenges, Proc. IEEE 93 (3) (2005) 682–691.
[7] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, N. Podhorszki, Characterizing output bottlenecks in a supercomputer, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, IEEE Computer Society Press, Los Alamitos, CA, USA, 2012, pp. 8:1–8:11.
[8] K. Wang, A. Kulkarni, M. Lang, D. Arnold, I. Raicu, Exploring the design tradeoffs for extreme-scale high-performance computing system software (2015).
[9] K. Wang, A. Kulkarni, M. Lang, D. Arnold, I. Raicu, Using simulation to explore distributed key-value stores for extreme-scale system services, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013, p. 9.
[10] A. Gheorghe, et al., Overview of the IBM Blue Gene/P project, IBM J. Res. Dev. 52 (1/2) (2008) 199–220.
[11] Parallel I/O on Mira, http://www.alcf.anl.gov/files/parallell_io.pdf, Accessed September 5, 2014.
[12] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, M. Wingate, PLFS: a checkpoint filesystem for parallel applications, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009.
[13] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, F. Zheng, Datastager: Scalable data staging services for petascale applications, in: Proceedings of the Eighteenth ACM International Symposium on High Performance Distributed Computing, 2009.
[14] M. Dorier, G. Antoniu, F. Cappello, M. Snir, L. Orf, Damaris: How to efficiently leverage multicore parallelism to achieve scalable, jitter-free I/O, in: Cluster Computing (CLUSTER), 2012 IEEE International Conference on, 2012, pp. 155–163.
[15] X. Zhang, K. Davis, S. Jiang, IOrchestrator: Improving the performance of multi-node I/O systems via inter-server coordination, in: High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for, 2010, pp. 1–11.
[16] HDF5, http://www.hdfgroup.org/hdf5/doc/index.html, Accessed September 5, 2014.
[17] NetCDF, http://www.unidata.ucar.edu/software/netcdf, Accessed September 5, 2014.
[18] F. Chen, D.A. Koufaty, X. Zhang, Hystor: Making the best use of solid state drives in high performance storage systems, in: Proceedings of the International Conference on Supercomputing, ICS '11, ACM, New York, NY, USA, 2011, pp. 22–32.
[19] X. Zhang, K. Davis, S. Jiang, iTransformer: Using SSD to improve disk scheduling for high-performance I/O, in: Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, IEEE, 2012, pp. 715–726.
[20] 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, 2012, pp. 1–11.
[21] M. Dorier, G. Antoniu, R. Ross, D. Kimpe, S. Ibrahim, CALCioM: Mitigating I/O Interference in HPC Systems through Cross-Application Coordination, in: Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, 2014, pp. 155–164.
[22] Z. Zhou, X. Yang, D. Zhao, P. Rich, W. Tang, J. Wang, Z. Lan, I/O-aware job scheduling and bandwidth allocation for petascale computing systems, in: Cluster Computing (CLUSTER), 2015 IEEE International Conference on, 2015.
[23] Cobalt Resource Manager, https://trac.mcs.anl.gov/projects/cobalt, Accessed September 5, 2014.
[24] Mira, 10-petaflops IBM Blue Gene/Q system.
[25] W. Tang, Z. Lan, N. Desai, D. Buettner, Fault-aware, utility-based job scheduling on Blue Gene/P systems, in: IEEE International Conference on Cluster Computing and Workshops, 2009, CLUSTER '09., 2009, pp. 1–10.
[26] W. Tang, N. Desai, D. Buettner, Z. Lan, Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P, in: 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS), 2010, pp. 1–11.
[27] W. Tang, Z. Lan, N. Desai, D. Buettner, Y. Yu, Reducing fragmentation on torus-connected supercomputers, in: Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International, 2011, pp. 828–839.
[28] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, R. Ross, Understanding and improving computational science storage access through continuous characterization, ACM Trans. Storage (TOS) 7 (3) (2011) 8:1–8:26.
[29] J. Jones, B. Nitzberg, Scheduling for parallel supercomputing: a historical perspective of achievable utilization, in: Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, 1659, 1999, pp. 1–16.
[30] Y. Xu, Z. Zhou, W. Tang, X. Zheng, J. Wang, Z. Lan, Balancing job performance with system performance via locality-aware scheduling on torus-connected systems, in: Cluster Computing (CLUSTER), 2014 IEEE International Conference on, 2014, pp. 140–148.
[31] A. Bhatele, K. Mohror, S.H. Langer, K.E. Isaacs, There goes the neighborhood: Performance degradation due to nearby jobs, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC'13, ACM, New York, NY, USA, 2013, pp. 41:1–41:12.
[32] J. Lofstead, R. Ross, Insights for exascale IO apis from building a petascale IO api, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC'13, ACM, New York, NY, USA, 2013, pp. 87:1–87:12.
[33] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, 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, 2010, pp. 1–12.
[34] X. Zhang, K. Davis, S. Jiang, Opportunistic data-driven execution of parallel programs for efficient I/O services, in: Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE Twenty-Sixth International, 2012, pp. 330–341.
[35] Y. Hashimoto, K. Aida, Evaluation of performance degradation in HPC applications with VM consolidation, in: Proceedings of the 2012 Third International Conference on Networking and Computing, ICNC '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 273–277.

[36] D. Skinner, W. Kramer, Understanding the causes of performance variability in HPC workloads, in: Workload Characterization Symposium, 2005. Proceedings of the IEEE International, 2005, pp. 137–149.
[37] A. Uselton, M. Howison, N. Wright, D. Skinner, N. Keen, J. Shalf, K. Karavanic, L. Oliker, Parallel I/O performance: From events to ensembles, in: Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on, 2010, pp. 1–11.
[38] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, M. Snir, Scheduling the I/O of HPC applications under congestion, Research Report, LIP, 2014.