CCF Transactions on High Performance Computing (2019) 1:49–59
https://doi.org/10.1007/s42514-019-00005-9

REGULAR PAPER

Software-defined QoS for I/O in exascale computing

Yusheng Hua[1] · Xuanhua Shi[1] · Hai Jin[1] · Wei Liu[1] · Yan Jiang[1] · Yong Chen[2] · Ligang He[3]

Received: 29 September 2018 / Accepted: 1 March 2019 / Published online: 1 April 2019

Abstract
Supercomputers' capability is approaching the exascale level, which enables large computing systems to run more jobs concurrently. Since modern data-intensive scientific applications can sometimes produce millions of I/O requests per second, I/O systems always suffer from heavy workloads and impede the overall performance. How to allocate I/O resources and guarantee the QoS (Quality of Service) for each individual application is becoming an increasingly important question. In this paper, we propose SDQoS, a software-defined QoS framework with the token bucket algorithm, aiming to meet the I/O requirements of concurrent applications contending for the I/O resources and improve the overall performance of the I/O systems. Evaluation shows that SDQoS can effectively control the I/O bandwidth within a 5%–10% deviation and improve the performance by 20% in extreme cases.

Keywords QoS · I/O · exascale computing

1 Introduction

The overall performance of all supercomputers in Top 500 supercomputer list has exceeded one ExaFlops (Dongarra et al. 2018). Sunway TaihuLight, as the fastest supercomter in China, has the peak performance of 120 PetaFlops. IBM Summit, the most powerful supercomputer deployed at Oak Ridge National Laboratory (ORNL), has the peak performance of nearly 200 PetaFlops. With the development of fast processors, new accelerators and multicore chips, the computational performance of supercomputers exhibits a sustained growth. However, it has become a critical problem that how these superior computing resources cooperate well with relatively weak I/O subsystem as a powerful, collective HPC system.

As more and more scientific and commercial applications become data-intensive, the bottleneck of HPC systems has shifted from computing to I/O. Large amount of I/O requests generated by these applications cause massive concurrent accesses to limited shared storage resources, leading to the performance degradation due to the I/O interference (Carns et al. 2009). Recent studies attempt to address these problems from the point of inter-application coordination (Dorier et al. 2014), optimized I/O forwarding framework (Ali et al. 2009), burst buffer (Kougkas et al. 2016; Shi et al. 2017), etc. These ideas focus on improving the overall performance of the I/O

✉ Xuanhua Shi
xhshi@hust.edu.cn

Yusheng Hua
yshua@hust.edu.cn

Hai Jin
hjin@hust.edu.cn

Wei Liu
ccloude@hust.edu.cn

Yan Jiang
harryjy_hust@hust.edu.cn

Yong Chen
yong.chen@ttu.edu

Ligang He
liganghe@dcs.warwick.ac.uk

[1] National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Huazhong University of Science and Technology, Wuhan 430074, China

[2] Department of Computer Science, Texas Tech University, Lubbock, TX, USA

[3] Department of Computer Science, University of Warwick, Coventry, UK

system, not on the requirement of individual applications to some extent.

Nowadays, large-scale computing centers would rent their spare resources to customers. Therefore, from the perspectives of both users and the HPC system, it is helpful to provide a mechanism of guaranteeing users' quality of service (QoS). In other words, the jobs submitted by users can obtain the desired performance with reduced cost in time and money. From the perspective of an HPC system, the QoS mechanism should be able to provide the better management of I/O resources and improve the utilization of the whole system.

In order to deliver the QoS control of I/O in an HPC system, two main challenges have to be addressed. First, expressing and transmitting these specific I/O requirements in different I/O stages is challenging because the resources needed to process I/O requests in different stages are not the same. Moreover, the heterogeneous I/O architecture complicates the applications' I/O paths and increase the difficulty of QoS control. Second, in order to meet the I/O requirement of individual applications, changes will often have to be made to the original I/O process inevitably, which may slow down the overall I/O performance in system. We call this the control overhead of the QoS mechanisms. Since I/O resources are relatively deficient in an HPC system, a large control overhead may lead to the degradation of the overall performance. It means that a proper balance should be achieved between overhead and accuracy.

In this paper, we propose SDQoS, a software-defined QoS framework for the I/O bandwidth control in large-scale HPC systems. The key idea is to integrate software-defined components, including data plane and control plane, into the conventional HPC storage system and provide a fine-grained QoS for different applications, while improving the overall performance of the system. We carried out a proof-of-concept implementation of SDQoS with PVFS and demonstrate its functionality, performance and scalability. The contributions of our work are as follows:

- Investigate the state-of-the-art HPC systems and summarize the characteristics of their I/O subsystems, which provide the basis for the design of SDQoS.
- Propose SDQoS, a new software-defined framework using token-bucket mechanisms for bandwidth control and for guaranteeing applications' I/O requirements.
- Provide a set of MPI-based interfaces for users to tag I/O requests with customized information which is fundamental for I/O identification in a sharing environment.
- Demonstrate the effectiveness of SDQoS and show its potential benefit for the exascale systems.

2 Motivation and challenges

2.1 Analysis of pre-exascale I/O architecture

We are stepping into the exascale era with the advent of powerful supercomputing systems such as Sunway and Summit. Although these systems have not reached the exaFLOP range yet, the architectures of these "pre-exascale systems" are useful for us to understand and investigate various issues in exascale computing, such as job scheduling, network transportation, and parallel file system.

Since we aim to provide the QoS of I/O for exacale computing, it is necessary to conduct a comprehensive investigation of current HPC systems and projected exascale systems. In this study, we summarize our findings including their architectures and characteristics as below.

Summit. As the No.1 supercomputer in the latest Top500 list (Dongarra et al. 2018), Summit is designed for the workload of the United States Department of Energy (DoE) Office of Science and other domains such as natural science and engineering (Vazhkudai et al. 2018).

Each compute node in Summit has two IBM POWER9 (P9) CPUs with six NVIDIA Volta V100 GPUs and are connected with a Mellanox EDR IB network. Its total 4608 nodes on 256 racks provide nearly 200 petaflops peak performance with the power consumption of 13 MW approximately. Summit has 250PB file system storage with 2.5 TB/s bandwidth. In order to provide a burst buffer, each compute node has 1.6 TB NVMe-device with 2.1 GB/s write bandwidth and 5.8 GB/s read performance, which eventually makes up a burst buffer with the capacity of 7.4PB and the bandwidth of 9.7 TB/s for the whole system. Summit does not have a shared burst buffer pool but provide on-node local burst buffers instead.

Sierra. Sierra is currently ranked No.3 with a theoretical peak performance of 119 petaflops. Its primary functionality is to conduct simulations and provide uncertainty quantification of the issues related to the National Nuclear Security Administrations (NNSA) stewardship of the US nuclear stockpile.

The overall architecture of Sierra is similar to that of Summit but has a difference in the scale due to the characteristics of the workload mainly targeted by the system. Sierra also has two P9 CPUs per node but only has four NVIDIA Volta V100 GPUs which is two less than Summit. However, Sierra performs better when running specific workloads requiring more GPU-CPU bandwidth. This is because less GPUs per CPU means that one GPU can achieve higher bandwidth from a single socket. In terms of the I/O subsystem, Sierra has the file system storage capacity of 150PB and the bandwidth of 1.5 TB/s in total, and a on-node local burst buffer capacity of 6.9PB and the bandwidth of 9.1 TB/s.

Sunway TaihuLight. Sunway TaihuLight is currently ranked No.2 with a theoretical peak performance of 125 petaflops. Unlike the above two systems, Sunway is not built for specific workloads but providing computing resources for a variety of users in industry and academia, who sumbit diverse large-scale jobs, such as climate modeling, earth subsurface modeling and inversion, sky simulation, and phase-field simulation (Fu et al. 2016).

Sunway is an entirely custom-designed system using the many-core SW26010 processors made in China. The architecture of Sunway is similar to a traditional HPC system, in which all compute nodes share a unified storage pool. Sunway has 20PB total storage with the 288 GB/s bandwidth, which is much smaller than Summit and Sierra. It is worthy of noting that Sunway does not have a separate burst buffer layer. Instead, it has two parts of storage per node: one is a high-speed storage area that is fully composed by SSDs, while the other is a RAID system consisting of hard disks. This architecture can be regarded as burst buffer to some extent because the I/O requests can be transferred to different storage areas according to different demands. However, the high-speed areas are not typical burst buffers since the data stored there are permanent and can not be flushed.

Tianhe-2. Tianhe-2 is currently ranked No.4 with a theoretical peak performance of 100 petaflops after recent upgrade. It is the most popular HPC system in China since it adopts the Xeon series processors, which, compared to Sunway, allows the users to run their applications without much modification of the codes. Tianhe-2 has the 19TB storage with the 1TB/s bandwidth. The 3T burst buffer is shared among 64 nodes for accelerating the I/O access.

Aurora. Aurora [9] is considered as the most advanced supercomputer, with an option of upgrading to 450 petaflops. Unlike the above four pre-exascale systems, which have been in production use, Aurora will be delivered in 2020 and provide computing resources for a broad range of computing and data-centric workloads such as in transportation, biological science, and renewable energy domains.

Aurora allocates compute nodes with the NVMe devices as the local processing node temporal storage, which moves the I/O node storage to the compute node. Burst buffer storage is also used to provide faster checkpointing, quicker recovery and better application performance. Detailed specification of the I/O system have not been released yet, but expect to be larger and faster than existing systems.

2.2 Characteristics of pre-exascale and projected exascale I/O architectures

From studying and analyzing these five state-of-the-art HPC systems discussed above, we can observe that I/O subsystems of different HPC systems differ considerably. The HPC systems in China, such as Tianhe-2 and Sunway, have focused more on the compute nodes than the I/O subsystems, compared to other systems in terms of total storage capacity, aggregate I/O bandwidth, and burst buffer capabilities. Also, in the newer systems, such as Summit, the I/O subsystem is designed more carefully, especially in burst buffer, which has more local SSDs per node. Although these pre-exascale and future HPC systems distinct from each other, they have shed a light on what exscale systems will be like in the future. We are especially interested in I/O subsystems in such a system and draw a conclusion of three critical characteristics from the above-mentioned representative systems based on our investigation and findings.

Complicated hierarchy. In the early days, hard disk drives are the dominant component in a storage system. Although hard disk drives remain widely used these days, an increasing number of high-speed storage mediums appear and become critical components in HPC centers.

The I/O hierarchy in supercomputers clearly exhibits a trend of being more complicated due to the diverse devices and performance requirements from increasingly data-driven scientific discovery needs (Wadhwa et al. 2018). HPC systems tend to use high-speed devices to fill up the gap between massive computing capacity and slow I/O subsystems. We can observe from the aforementioned five latest supercomputers that SSDs can be used in two dimensions: (1) horizontal. In this type of architecture, SSDs are used as the alternative storage resources to disks. From an application's perspective, the I/O requests with higher demand for read/write bandwidth can be directly forwarded to SSDs. Since the total capacity of SSDs are not always adequate, it is important to design effective strategies as to which type of applications should use SSDs more often. Also, it is easier for the design of file systems with placing SSD and HDD horizontally because both devices are exposed to the upper file system through a uniform view. The target devices of the I/O requests can be determined in a single file system. (2) vertical. In this type of architecture, I/O subsystem appears to be deeper that the NVEe devices, or the common SSDs are added between the compute nodes and normal disk storage, from which scientific applications with burst I/O can benefit significantly. The function of these devices is similar to the memory cache systems to some extent. However, the storage capacity is much larger than the memory. The network between these storage devices is also slower and often becomes the bottleneck. The file system design becomes complicated too with the vertical perspective, since we need another light-weight file system running in the burst buffer layer to receive the I/O requests from the upper layer and forward to the target servers. For example, Summit has a two-tiered I/O subsystem with separate name spaces for burst buffer and parallel file system.

Hybrid burst buffer. Burst buffer (Liu et al. 2012) is an indispensable part of current HPC systems in order to

alleviate the problem of slow I/O. It can greatly enhance the I/O performance thanks to its full SSD composition. Common burst buffers can be classified into two types: (1) shared burst buffer. Shared burst buffer has a large chunk of global high-speed storage which can be accessed by all compute nodes (or several nodes share a burst buffer block). This kind of architecture can support the "N-1" I/O access pattern well (i.e., *N* processes writing to a single, shared file)) (Vazhkudai et al. 2018). (2) local burst buffer. local burst buffer (always consisting of NVM devices) is always attached to a compute node. This solution greatly benefit the applications performing N–N I/O access patterns (i.e., each processes writing to a different file, e.g, checkpoint Bent et al. 2009).

High concurrency. In HPC systems, a parallel file system (PFS) is always shared by numerous applications or processes to provide large I/O bandwidth and highly concurrent accesses. However, with the rapid growth of computing power, PFS suffers from increasingly heavy workloads and cannot provide sufficient bandwidth for every single application (Thapaliya et al. 2014). The situation will worsen in the exascale systems, in which even more jobs will run simultaneously.

Based on our studies and analyses, we have concrete reasons to believe that the upcoming exascale I/O subsystems will have more complicated hierarchy, with the hybrid burst buffer systems to deal with challenges posed by more simultaneously running I/O workloads.

2.3 QoS in exascale computing

QoS is a widely discussed topic in areas such as networking and cloud computing. However, few studies exist for investigating the QoS in supercomputers because in the early days, HPC systems were built for specific usages and only for limited users to run their customized jobs. However, with the rapid growth of computing power and considerable reduction of the cost, HPC centers tend to offer a "utility" computing model too, with renting spare computing resources for higher resource utilization and the economic efficiency. We believe this situation will remain in the upcoming exascale era due to their much more powerful computing resources and more potential users. Next, we will elaborate the challenges of meeting QoS in upcoming exascale computing era.

Challenges on semantic interpretation of QoS. As discussed above, future exascale systems will most likely have more heterogeneous architecture, which complicates applications' I/O paths. I/O requests may go through various devices and be processed by different software, which leads to difficulty to control the I/O requirements. For example, if two jobs both require 1 GB/s bandwidth and we throttle the I/O requests at the metadata regardless of their various I/O paths, the performance of these two applications may vary significantly. Thus, applications' QoS should be achieved through the cooperation of all I/O components. Furthermore, an elegant method is needed to translate one single I/O requirement into specific parameters in different components, depending on their characteristics. For example, a specified I/O bandwidth requirement can mean I/O per second in the metadata server, but means packages per second in network transfer.

Challenges on reducing control overhead. The issue of I/O interference caused by high concurrency has been extensively discussed. Previous studies (Dorier et al. 2014; Ali et al. 2009; Kougkas et al. 2016; Shi et al. 2017) often tend to make a global improvement while ignoring the demand of individual applications. However, when satisfying the QoS of individual applications becomes a new demand, how to guarantee the I/O requirements of applications and at the same time alleviate I/O contention becomes a critical issue. According to our experiences, QoS control can sometimes lead to more overhead such as lock contentions when scheduling queues. Thus, we should make careful considerations when implementing QoS functionality and minimize the overhead.

3 Software-defined QoS design and implementation

"Software-defined" is a well-known concept that separates control component from data flow for achieving feasible and fine-grained management. We believe that software-defined approaches can benefit HPC I/O because its resource management mechanisms are exactly what needed for more complicated I/O environment in the upcoming exascale systems.

There are two advantages we believe the software-defined approach can offer for meeting QoS in HPC: (1) flexible policy. Typically, like Lustre, file system configurations are set manually to meet the I/O requirements. These commands rely on the administrators' experiences and cannot adjust to the dynamic requirements at real time. Since I/O behaviors in HPC are fluctuate and sometimes bursting, with the adoption of a global control plane, users can customize more suitable and integrated policies, which can be enforced to the involved I/O components and take effect when a pre-defined condition is met. Also, through this way, an I/O requirement can be interpreted into different policies in different components, which ensures the correctness of the control result and unifies the semantic interpretation of QoS. (2) global view. One key idea of software-defined concept is to separate a solo control plane to monitor global I/O behaviors and enforce the optimized policies. In hybrid HPC architectures, there are many devices and their corresponding softwares working together to maintain the regular work of the whole system. However, relations between these components

may not be that close and I/O requests may be offered with various services due to different schemes or statuses of each component. For example, if an application gets enough I/O bandwidth form burst buffer, it will need less I/O resources from PFS to meet its requirement. Therefore, a global view offered by the software-defined approach is necessary and makes it easy to combine these components together for a unified management.

Since the software-defined concept is a useful tool for implementing I/O QoS in HPC, we adopt the idea and implement it in PVFS, the detail of which will be introduced in later sections.

3.1 SDQoS architecture

We show our proposed SDQoS architecture in Fig 1. SDQoS works with a traditional HPC architecture which includes compute nodes for running applications, network for data movement, metadata server and storage server for providing metadata and data access. We aim to show that the SDQoS framework is suitable for general HPC systems corresponding to this architecture.

In the SDQoS design, we consider three important details, which are fundamental for the implementation of critical components: (1) We try to make the QoS control as transparent as possible, which means that users can specify a desired bandwidth of their applications with few modifications of their codes. To achieve this goal, we modify PVFS at the server side, which is away from the upper application layer and is straightforward for implementing the bandwidth control. For convenience, we also modify MPI so that requirements of different applications can be assigned when users summit their jobs by appending *mpirun* orders with *-bandwidth* argument. These I/O requirements will pass through the whole I/O stack along with the corresponding I/O requests. (2) Since PVFS does not have the owner information of received I/O requests, it is hard to identity I/O requests from different applications. Due to the fact that high performance applications usually use MPI interfaces to raise I/O requests, we customize another set of interfaces to be an alternative, in order to tag the I/O requests of different applications with any necessary information (eg. pid). These tags are first set in *mpirun*, and jobs can get these tags when using our specified interfaces (eg. *MPI_FILE_WRITE_ud*). Then, although the single application's jobs are running on different nodes, they can have the same tags. These tags will be encapsulated into the I/O request structure in PVFS side and provide feasibility for the following I/O classification and QoS control. (3) We try to abstract the I/O resources in a proper manner. Since the I/O flow in HPC is similar to network packages, we can apply the controlling algorithms in the network area to SDQoS. While being inspired by Lustre's QoS implementation, we choose the token bucket as the cornerstone of our design. The token-bucket algorithm is helpful for traffic shaping and rate limiting. It not only provides SDQoS with the capability for handling burst traffic, but also guarantees the basic requirements of I/O bandwidth.

In order to realize the concept of software-defined QoS, we have also designed the specific data plane and the control plane, as explained in detail below.

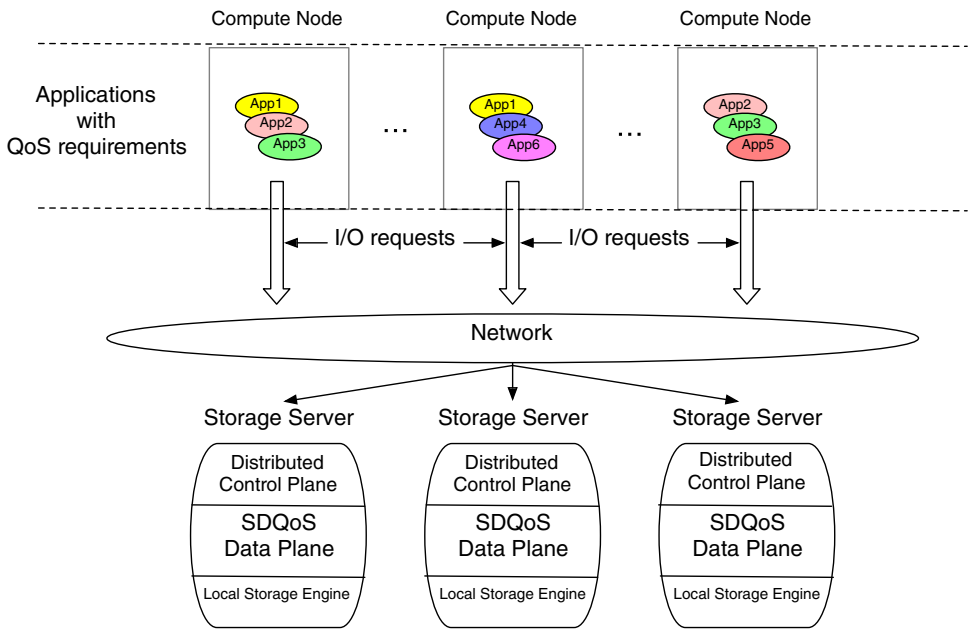


Fig. 1 SDQoS overall architecture

3.2 SDQoS control plane

Control plane is the most critical part in our SDQoS design. It should provide the global view and formulate flexible policies for different I/O requirements. We carefully consider the specific characteristics of HPC I/O and propose a new, more suitable control plane design.

A typical control plane is in charge of scheduling computation, load balancing, and recovering from failures (Qu et al. 2018). Since these functions have been provided by other components in HPC systems, in SDQoS, we simplify the design and make the control plane a logical component, without occupying a dedicated, physical server, but is distributed among all storage nodes involving the I/O processing, which can reduce the communication overhead between data planes and the control plane.

SDQoS makes policies based on the token bucket algorithm. The control plane maintains the token buckets of all applications and generating the tokens at real time. It provides interfaces for the I/O requests to check the left tokens and make sure whether it can be processed immediately or buffered in the waiting queue. For example, if we get an incoming I/O request's size (eg. 32 K or 64 K), then the tokens per I/O request needed for processing can be calculated as $t = \frac{s}{1M}$ under the assumption that one token represents 1 MB/s bandwidth. Here, s represents the request size. Assuming that App1 has the bandwidth requirement of 100 MB/s and its request size is 64 K. Then, the token bucket of App1 will generate 100 tokens per second and only when the corresponding token bucket has no less than 0.0625 tokens can an I/O request be processed at once. For different bandwidth requirements from different applications, we configure their token buckets respectively according to the above procedure. Since the I/O requests of different applications always saturate the storage servers and show the trend of being disordered, tokens of those I/O requests which have relatively larger intervals can be accumulated and allow the servers to handle the potential burst I/O.

3.3 SDQoS data plane

The SDQoS Data Plane is distributed across all storage nodes and resides in the local storage engine. It is in charge of pre-processing I/O requests for classification and throttling operations. Since it needs to concurrently process large amounts of I/O requests, a well optimized design is needed to alleviate the I/O contention and the control overhead.

We implemented the data plane component at the PVFS server side and combined it with the control plane to perform the precise bandwidth control. First, it learns applications' I/O bandwidth requirements from the control plane. Then it classifies all incoming I/O requests based on the tag and buffers them in the corresponding queues. In order to explain our implementation, it is necessary to introduce the I/O processing procedure in PVFS.

PVFS is an open source parallel file system, which distributes the file data across multiple servers and provides high performance access for parallel applications. We identify its I/O procedure by analyzing its source code: when an I/O request arrives, the file system calls the **issue_or_delay_io_operation()** function to determine whether it can be processed immediately (PVFS can only processes 16 I/O requests concurrently). If an I/O request is blocked, it will be buffered in a uniform queue and wait to be processed. The authentic function to deal with the I/O request is **lio_listio()**. After the accomplishment of one I/O request, it spawns a thread to deal with the blocked I/O requests asynchronously. Therefore, from the procedure we can see that PVFS only have one queue for buffering and all I/O requests are mixed together in the file system. In order to perform further management, we have to first create separate queues to be used by different applications exclusively. Redesigning the queueing mechanisms in file systems needs careful consideration and integration with the original I/O handling procedure. We redesign the queueing mechanism by creating a new queue upon the arriving of I/O requests of a new application. Besides the initial judgment condition, we will block the I/O requests if their corresponding token buckets do not have enough tokens.

4 Evaluation

In this section, we will present the evaluation of our SDQoS in terms of functionality, performance, overhead and scalability. SDQoS is a general software-defined framework for HPC systems and can be implemented in various file systems. We chose PVFS as the platform for current implementation and evaluation in this study.

4.1 Evaluation setup

The specific environment of our evaluation is shown in Table 1. We used 8 nodes in total and deployed the modified PVFS on all of them. IOR is used as the benchmark and to simulate applications' I/O behaviors.

4.2 Functionality evaluation

Functionality is the most important part of our evaluation because the core principle of the QoS control is to enable the applications to obtain their desired resources (more specifically bandwidth in this test).

The evaluation in this subsection is to verify the effectiveness of our PVFS-based SDQoS and measure the deviation between the preset desired bandwidth and the actual

Table 1 Evaluation environment

CPU	SW6A @ 1.60 GHz with 16 cores
Memory	128 GB
Connection	Gigabit Ethernet (1000Mbps)
Storage	Toshiba model MBF2300RC 300 GB
OS	linux-based OS with 4.4.15-aere+ kernel
Filesystem	orangefs-2.9.3
Test tool	IOR-2.10.1

Table 2 Single application bandwidth control

Desired Bandwdth (MB/s)	Actual Bandwidth (MB/s)	Deviation (%)
560	513.47	8.3
480	458.65	4.44
240	217.82	9.24

bandwidth that the applications acquire after the management. The evaluation is divided into two parts: (1) first, we ran a single application with different desired bandwidths to verify if the basic functionality works; (2) second, we ran two applications with different desired bandwidths to verify if SDQoS works and observe how they influence each other.

In order to set the proper bandwidth for the application, we first tested the server's I/O capability using IOR and turned off the QoS switch. It shows that the aggregate bandwidth of 8 servers is 535.82 MB/s.

Table 2 shows the actual bandwidth of a single application under different configurations. It can be seen that the actual bandwidth is very close to the desired bandwidth with the deviation of less than 10%.

Table 3 shows the results of two applications running simultaneously with the QoS control. The result shows that each application can get the bandwidth close to the desired bandwidth when multiple jobs are contending for the I/O resources, which means that our SDQoS can work well in the multitasking environment. The data in tables are the average result of five independent runs with small deviations between them. Since the applications' I/O requests are always uncertain and erratic when arriving at the filesystem, the stability of our results can show that the adopted token bucket algorithm is effective in I/O flow controlling.

It can be concluded from the evaluation results that SDQoS have the ability to ensure applications' various I/O requirements with accurate and stable performance. We also find some clues about why SDQoS have the deviation between actual and desired bandwidth and how to mitigate it. We thought that besides the fluctuation of disks and network, locking overhead is another important factor due to the fact that scheduling between multiple queues (which are created by us for classification) generates more lock/unlock operations and contributes to the deviation. In this work, we only decrease the granularity of the locks as far as we can. But we believe that finding more optimized approaches to reduce the deviation is feasible and meaningful.

4.3 Performance evaluation

The evaluation in this subsection aims to prove that SDQoS can improve the overall performance under the extreme I/O scenario when the I/O interference between applications severely degrades the I/O performance. The IOR configurations are shown in Table 4 and test results are shown in Table 5. We can see the performance improvement of 22% after using SDQoS.

We suppose that the main reason of the I/O performance decrease under the I/O interference scenario is that servers

Table 4 IOR configuration of performance evalutaion

	APP1	APP2
Request size (KB)	64	4
Total amount (GB)	8	8
Thread number	16	16
Access pattern	Strided	Random

Table 5 Performance improvement of SDQoS

	Aggregated bandwidth (MB/s)	Performance improvement (%)
QoS OFF	299.77	22
QoS ON	366.64	

Table 3 Multiple applications bandwidth control

APP1			APP2		
Desired bandwidth (MB/s)	Actual bandwidth (MB/s)	Deviation (%)	Desired bandwidth (MB/s)	Actual bandwidth (MB/s)	Deviation (%)
320	294.32	8.03	80	73.11	8.61
240	216.74	9.70	160	144.79	9.51
400	361.66	9.59	80	72.76	9.05

eventually process the I/O requests in a serial manner regardless of the degree of parallelism of the applications. So when the large I/O requests are mixed with small I/O requests, the latter will have more chances to block the former and generate the large overhead. Our SDQoS can limit the token rate of small I/O requests which can be seen as a delay operation and use the extra queue management scheme which buffers the blocked requests and make it be processed at a proper time, so as to alleviate the contention. In this case, we make the I/O requests of 64 K have more chances to be served by limiting the token rate of the I/O requests of 4 K, which effectively results in overall performance improvement.

4.4 Overhead and scalability evaluation

Scalability is a critical issue in distributed filesystems, we have to make sure that our design of SDQoS will not jeopardize the good scalability of the original PVFS and can work with the increasing system scale at low overhead. So In this subsection, we mainly discuss the overhead of SDQoS and its scalability.

We exposed the overhead of SDQoS by turning on and off the QoS control without any specific I/O requirements. So the I/O requests will only go through all the components of SDQoS without being processed. This way, the overhead mainly composes of logical judgement, queue maintenance and thread creating. The experiment has been conducted with different numbers of nodes (1, 2, 4, 8) and the results are shown in Table 6. It can be seen that the overhead is very low and increases slowly under various number of nodes, which mainly owes to the decoupled and distributed design of SDQoS.

5 Related work

The QoS control has been extensively studied in network (Altmann et al. 2002; Houngbadji and Pierre 2010; Clark et al. 1992; Sharma et al. 2012) and the token bucket algorithm is a effective tool for network package management(Sahu et al. 2000; Tang and Tai 1999). Despite the similarities between the I/O flow and the network flow, the challenges arise as to the situation where that I/O requests can be simply discarded when the I/O systems are saturated. It motivates us to adapt the token bucket algorithm to a more suitable form for HPC I/O.

As mentioned above, the QoS control should have a holistic consideration of the whole I/O procedures from providing proper interfaces for users to managing I/O resources in an optimized way. Many works have been conducted to resolve specific issues involving the procedure. (Wijayaratne and Reddy 1999; Bruno et al. 1999) provide the QoS management for disk I/O. (Rajachandrasekar et al. 2012) focuses on minimizing network contention in parallel filesystems. (Gildfind and McDonell 2009; Houngbadji and Pierre 2010) implement the bandwidth control in shared distributed storage systems. Lustre(Schwan 2003) and Ceph(Weil et al. 2006) also provide the bandwidth control mechanisms. Qian et al. (2017) integrated the QoS function into Lustre based on the token-bucket algorithm which inspired us a lot. But we provide a more complete QoS design on PVFS in our work.

A software-defined approach to implementing the QoS control in HPC is another highlight of our work. Similar works have been carried out in cloud computing. IOFLOW(Thereska et al. 2013) (known as sRoute(Stefanovici et al. 2016) now) and Crystal(Gracia-Tinedo et al. 2017) are representative works that provide the ability to track, schedule, and control the I/O path in the cloud environment. However, the centralized design of the control plane can be a bottleneck for high scalability in HPC. Compared to them, SDQoS proposes a lightweight decentralized control plane design, which shows the promising advantage for exascale computing.

Table 6 Overhead under increasing scale

Node Number	1	2	4	8
Aggregate bandwidth with QoS OFF (MB/s)	118.95	238.76	399.70	538.14
Aggregate bandwidth with QoS ON (MB/s)	118.31	235.76	393.63	523.38
Overhead (%)	0.54	1.27	1.54	2.7

6 Conclusion and future work

Exascale HPC systems is appearing with more high-speed devices and hybrid architectures, which will provide tremendous computing resources for scientific or commercial applications. However, the gap between computing and I/O is becoming bigger. Due to the trend of multi-tenant scenario and more intense I/O contention, a sophisticated scheme to provide QoS for users who share the I/O resources becomes increasingly critical. In this paper, we investigate the HPC I/O characteristics from a number of popular HPC systems and propose SDQoS, a software-defined QoS framework, which provides the QoS mechanisms for large scale computing centers. The evaluation shows that SDQoS can well control the I/O bandwidth and improve the performance under severe I/O environments because of the optimized management using the token bucket algorithm. We believe that the SDQoS-like components will be an indispensable part in future HPC systems, which will benefit the users

and the administrators. In future, we will continue to further optimize SDQoS to make it suitable for more complicated architectures, and also conduct further research on the I/O interference issue during the QoS management process.

Acknowledgements The work is supported by the National Key R&D Program of China(No. 2017YFC0803700), NSFC (No. 61772218, 61433019), and the Outstanding Youth Foundation of Hubei Province (No.2016CFA032).

References

Altmann, J., Daanen, H., Oliver, H., Suarez, A.-B.: How to market-manage a qos network. In: Proceedings of 21st annual joint conference of the IEEE computer and communications societies (INFOCOM), vol. 1, pp. 284–293, IEEE, (2002)

Ali, N., Carns, P., Iskra, K., Kimpe, D., Lang, S., Latham, R., Sadayappan, P.: Scalable I/O forwarding framework for high-performance computing systems. In: Proceedings of IEEE international conference on cluster computing, pp. 1–10, IEEE, (2009)

Argonne national laboratory's aurora system. https://www.intel.cn/content/www/cn/zh/high-performance-computing/intel-argonne-aurora-announcement-presentation.html. Accessed 10 Sept 2018

Bruno, J., Brustoloni, J., Gabber, E., Ozden, B., Silberschatz, A.: Disk scheduling with quality of service guarantees. In: Proceedings of IEEE international conference on multimedia computing and systems, vol. 2, pp. 400–405, IEEE, (1999)

Bent, J., Gibson, G., Grider, G., McClelland, B., Nowoczynski, P., Nunez, J., Polte, M., Wingate, M.: Plfs: a checkpoint filesystem for parallel applications. In: Proceedings of the international conference on high performance computing, networking, storage and analysis, pp. 21, ACM, (2009)

Clark, D. D., Shenker, S., Zhang, L.: Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In: Proceedings of the conference on communications architecture and protocols, pp. 14–26, ACM, (1992)

Carns, P., Latham, R., Ross, R., Iskra, K., Lang, S., Riley, K.: 24/7 characterization of petascale I/O workloads. In: Proceedings of IEEE international conference on cluster computing, pp. 1–10, IEEE, (2009)

Dorier, M., Antoniu, G., Ross, R., Kimpe, D., Ibrahim, S.: CALCioM: mitigating I/O interference in HPC systems through cross-application coordination. In: Proceedings of 28th IEEE international parallel and distributed processing symposium, pp. 155–164, IEEE, (2014)

Dongarra, J., Meuer, H., Strohmaier, E.: Top500 supercomputing sites. http://www.top500.org. Accessed 10 Sept 2018

Fu, H., Liao, J., Yang, J., Wang, L., Song, Z., Huang, X., Yang, C., Xue, W., Liu, F., Qiao, F., Zhao, W., Yin, X., Hou, C., Zhang, C., Ge, W., Zhang, J., Wang, Y., Zhou, C., Yang, G.: The sunway taihulight supercomputer: system and applications. Sci. China Inform. Sci. **59**(7), 072001 (2016)

Gildfind, A. J. A., McDonell, K. J.: "Method for empirically determining a qualified bandwidth of file storage for a shared filed system," Sept. 15 . US Patent 7,590,775 (2009)

Gracia-Tinedo, R., Sampé, J., Zamora, E., Sánchez-Artigas, M., García-López, P., Moatti, Y., Rom, E.: Crystal: software-defined storage for multi-tenant object stores. In: Proceedings of the 15th Usenix conference on file and storage technologies, pp. 243–256, USENIX Association, (2017)

Houngbadji, T., Pierre, S.: Qosnet: an integrated qos network for routing protocols in large scale wireless sensor networks. Comput. Commun. **33**(11), 1334–1342 (2010)

Kougkas, A., Dorier, M., Latham, R., Ross, R., Sun, X. H.: Leveraging burst buffer coordination to prevent I/O interference. In: Proceedings of 12th IEEE international conference on e-science, pp. 371–380, IEEE, (2016)

Liu, N., Cope, J., Carns, P. Carothers, C., Ross, R., Grider, G., Crume, A., Maltzahn, C.: On the role of burst buffers in leadership-class storage systems, In: Proceedings of 28th IEEE symposium on mass storage systems and technologies (MSST), pp. 1–11, IEEE, (2012)

Qian, Y., Li, X., Ihara, S., Zeng, L., Kaiser, J., Süß, T., Brinkmann, A.: A configurable rule based classful token bucket filter network request scheduler for the lustre file system. In: Proceedings of the international conference on high performance computing, networking, storage and analysis, pp. 6, ACM, (2017)

Qu, H., Mashayekhi, O., Shah, C., Levis, P.: Decoupling the control plane from program control flow for flexibility and performance in cloud computing. In: Proceedings of the thirteenth euroSys conference, pp. 1, ACM, (2018)

Rajachandrasekar, R., Jaswani, J., Subramoni, H., Panda, D. K.: Minimizing network contention in infiniband clusters with a qos-aware data-staging framework. In: Proceedings of IEEE international conference on cluster computing, pp. 329–336, IEEE, (2012)

Schwan, P.: Lustre: building a file system for 1000-node clusters. In: Proceedings of the 2003 linux symposium, vol. 2003, pp. 380–386, (2003)

Sahu, S., Nain, P., Diot, C., Firoiu, V., Towsley, D.: "On achievable service differentiation with token bucket marking for tcp," In: Proceedings of ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pp. 23–33, ACM, (2000)

Sharma, S., Katramatos, D., Yu, D., Shi, L.: Design and implementation of an intelligent end-to-end network QoS system. In: Proceedings of the international conference on high performance computing, networking, storage and analysis, pp. 1–11, ACM, (2012)

Shi, X., Li, M., Liu, W., Jin, H., Yu, C., Chen, Y.: SSDUP: a traffic-aware ssd burst buffer for HPC systems. In: Proceedings of the international conference on supercomputing, pp. 27, ACM, (2017)

Stefanovici, I. A., Schroeder, B., O'Shea, G., Thereska, E.: sroute: treating the storage stack like a network. In: Proceedings of the 14th Usenix conference on file and storage technologies, pp. 197–212, USENIX association, (2016)

Tang, P. P., Tai, T. Y.: Network traffic characterization using token bucket model. In: Proceedings of 18th annual joint conference of the IEEE computer and communications societies (INFOCOM), pp. 51–62, IEEE, (1999)

Thereska, E., Ballani, H., O'Shea, G., Karagiannis, T., Rowstron, A., Talpey, T., Black, R., Zhu, T.: Ioflow: a software-defined storage architecture. In: Proceedings of the 24th ACM symposium on operating systems principles, pp. 182–196, ACM, (2013)

Thapaliya, S., Bangalore, P., Lofstead, J., Mohror, K., Moody, A.: Iocop: managing concurrent accesses to shared parallel file system. In: Proceedings of 43rd IEEE international conference on parallel processing workshops (ICCPW), pp. 52–60, IEEE, (2014)

Vazhkudai, S., de Supinski, B., Bland, A., Geist, A., Sexton, J.,Kahle, J., Zimmer, C., Atchley, S., Oral, S., Maxwell, D., VergaraLarrea, V., Bertsch, A., Goldstone, R., Joubert, W., Chambreau, C.,Appelhans, D., Blackmore, R., Casses, B., Chochia, G., Davison, G.,Ezell, M., Gooding, T., Gonsiorowski, E., Grinberg, L., Hanson, B.,Hartner, B., Karlin, I., Leininger, M., Leverman, D., Marroquin, C.,Moody, A., Ohmacht, M., Pankajakshan, R., Pizzano, F., Rogers, J.,Rosenburg, B., Schmidt, D., Shankar, M., Wang, F., Watson, P.,Walkup, B., Weems, L., Yin, J.: The design, deployment, andevaluation of the coral pre-exascale systems. In Proceedings of the international conference on high performance computing, networking, storage and analysis, pp. 52, IEEE, (2018)

Wijayaratne, R., Reddy, A. N.: Integrated QOS management for disk I/O. In: Proceedings of IEEE international conference on multimedia computing and systems, vol. 1, pp. 487–492, IEEE, (1999)
Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D., Maltzahn, C.: Ceph: a scalable, high-performance distributed file system. In: Proceedings of the 7th symposium on operating systems design and implementation, pp. 307–320, USENIX Association, (2006)
Wadhwa, B., Byna, S., Butt, A.R.: Toward transparent data management in multi-layer storage hierarchy of hpc systems, In: Proceedings of IEEE international conference on cloud engineering, pp. 211–217, IEEE, (2018)
Yildiz, O., Dorier, M., Ibrahim, S., Ross, R., Antoniu, G.: On the Root Causes of Cross-Application I/O Interference in HPC Storage Systems. In: Proceedings of IEEE international parallel and distributed processing dymposium, pp. 750–759, IEEE, (2016)

Yusheng Hua received the bachelor degree in computer science and technology from Huazhong University of Science and Technology, China, in 2017. He is currently a PhD candidate at National Engineering Research Center for Big Data Technology and System Services Computing Technology and System/ Services Computing Technology and System Lab, Huazhong University of Science and Technology, China. His research interests focus on the HPC I/O, HPC QoS and parallel filesystem.

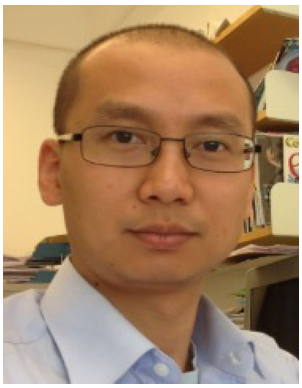
Xuanhua Shi is a professor in National Engineering Research Center for Big Data Technology and System Services Computing Technology and System/ Services Computing Technology and System Lab, Huazhong University of Science and Technology (China). He received the PhD degree in computer engineering from Huazhong University of Science and Technology, China, in 2005. From 2006, he worked as an INRIA Post-Doc in PARIS team at Rennes for one year. His current research interests focus on the cloud computing and big data processing. He published over 100 peer-reviewed publications, received research support from a variety of governmental and industrial organizations, such as National Science Foundation of China, Ministry of Science and Technology, Ministry of Education, European Union, Alibaba, ByteDance, Intel and so on. Shi is a senior member of IEEE and CCF.

Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. He was Dean of the School of Computer Science and Technology at HUST from 2005 to 2014. Jin received his PhD in computer engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientist of National 973 Basic Research Program Project of Virtualization Technology of Computing System. Jin was the member of Grid Forum Steering Group (GFSG). He has co-authored 15 books and published over 400 research papers. His research interests include computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage, and network security. Jin is named the steering committee chair of several International Conferences, such as GPC, APSCC, FCST, and ChinaGrid. Jin is a member of the steering committee of CCGrid, NPC, GCC, ATC, and UIC. Jin is a fellow of the IEEE and a member of the ACM.

Wei Liu received the bachelor degree in computer science and technology from Huazhong University of Science and Technology, China, in 2016. He is a master student at National Engineering Research Center for Big Data Technology and System Services Computing Technology and System/ Services Computing Technology and System Lab, Huazhong University of Science and Technology, China. His current research interests focus on the HPC I/O, cloud computing and parallel filesystem.

Yan Jiang received the bachelor degree in computer science and technology from Huazhong University of Science and Technology, China, in 2017. He is a master student at National Engineering Research Center for Big Data Technology and System Services Computing Technology and System/ Services Computing Technology and System Lab, Huazhong University of Science and Technology, China. His current research interests focus on the HPC I/O, burst buffer and parallel filesystem.

Yong Chen is an Associate Professor and the Director of the Data-Intensive Scalable Computing Laboratory in the Computer Science Department of Texas Tech University. He is also the Site Director of the Cloud and Autonomic Computing site at Texas Tech University. His research interests include data-intensive computing, parallel and distributed computing, high-performance computing, computer systems, and cloud computing. More information about him can be found from: http://www.myweb.ttu.edu/yonchen/.



Ligang He ireceived the bachelor's and master's degrees from the Huazhong University of Science and Technology, Wuhan, China, and the PhD degree in computer science from the University of Warwick, United Kingdom. He was also a postdoctoral researcher at the University of Cambridge, United Kingdom. In 2006, he joined the Department of Computer Science at the University of Warwick as an assistant professor, and then became an associate professor. His areas of interest are parallel and distributed computing, high-performance computing, and cloud computing. He is a member of the IEEE.