Automatic, Application-Aware I/O Forwarding Resource Allocation

Xu Ji, *Tsinghua University; National Supercomputing Center in Wuxi;* Bin Yang and Tianyu Zhang, *National Supercomputing Center in Wuxi; Shandong University;* Xiaosong Ma, *Qatar Computing Research Institute, HBKU;* Xiupeng Zhu, *National Supercomputing Center in Wuxi; Shandong University;* Xiyang Wang, *National Supercomputing Center in Wuxi;* Nosayba El-Sayed, *Emory University;* Jidong Zhai, *Tsinghua University;* Weiguo Liu, *National Supercomputing Center in Wuxi; Shandong University;* Wei Xue, *Tsinghua University; National Supercomputing Center in Wuxi*

https://www.usenix.org/conference/fast19/presentation/ji

This paper is included in the Proceedings of the
17th USENIX Conference on File and Storage Technologies (FAST '19).

February 25–28, 2019 • Boston, MA, USA

978-1-939133-09-0

Automatic, Application-Aware I/O Forwarding Resource Allocation

Xu Ji[1,2], Bin Yang[2,3], Tianyu Zhang[2,3], Xiaosong Ma[4], Xiupeng Zhu[2,3], Xiyang Wang[3], Nosayba El-Sayed[*] [5], Jidong Zhai[1], Weiguo Liu[2,3], and Wei Xue[†] [1,2]

[1]Tsinghua University, [2]National Supercomputing Center in Wuxi, [3]Shandong University, [4]Qatar Computing Research Institute, HBKU, [5]Emory University

Abstract

The I/O forwarding architecture is widely adopted on modern supercomputers, with a layer of intermediate nodes sitting between the many compute nodes and backend storage nodes. This allows compute nodes to run more efficiently and stably with a leaner OS, offloads I/O coordination and communication with backend from the compute nodes, maintains less concurrent connections to storage systems, and provides additional resources for effective caching, prefetching, write buffering, and I/O aggregation. However, with many existing machines, these forwarding nodes are assigned to serve a fixed set of compute nodes.

We explore an automatic mechanism, DFRA, for application-adaptive *dynamic forwarding resource allocation*. We use I/O monitoring data that proves affordable to acquire in real time and maintain for long-term history analysis. Upon each job's dispatch, DFRA conducts a history-based study to determine whether the job should be granted more forwarding resources or given dedicated forwarding nodes. Such customized I/O forwarding lets the small fraction of I/O-intensive applications achieve higher I/O performance and scalability, meanwhile effectively isolating disruptive I/O activities. We implemented, evaluated, and deployed DFRA on Sunway TaihuLight, the current No.3 supercomputer in the world. It improves applications' I/O performance by up to 18.9×, eliminates most of the inter-application I/O interference, and has saved over 200 million of core-hours during its test deployment on TaihuLight for 11 months. Finally, our proposed DFRA design is not platform-dependent, making it applicable to the management of existing and future I/O forwarding or burst buffer resources.

1 Introduction

Supercomputers today typically organize the many components of their storage infrastructure into a parallel and global controlled file system (PFS). Performance optimization by manipulating the many concurrent devices featuring different performance characteristics is a complicated yet critical task to administrators, application developers, and users. Moreover, it gets more challenging due to I/O contention and performance interference caused by concurrent jobs sharing the same PFS, bringing significant I/O performance fluctuation [28, 38, 40, 44, 61]. Meanwhile, different applications have vastly different I/O demands and behaviors, making it impossible for center administrators to decide one-size-for-all I/O configurations.

The task is even more difficult when it comes to the design and procurement of future systems. It is hard for machine owners to gauge the I/O demand from future users and design a "balanced" system with coordinated computation, network, and I/O resources. In particular, design and procurement typically happen years before any application could test run, while even decades-old programs usually see very different performance and scalability due to newer architecture/hardware/software on the more powerful incoming machine.

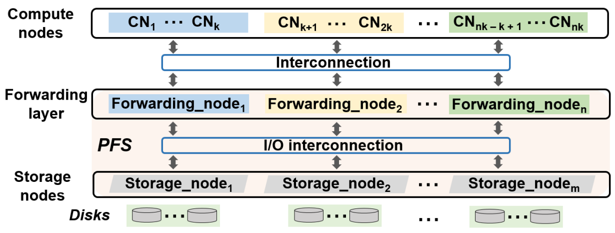


Figure 1: Typical I/O forwarding architecture for supercomputers

To give an example, consider the design of an *I/O forwarding infrastructure* [19], a widely adopted I/O subsystem organization that adds an extra forwarding layer between the compute nodes and storage nodes, as illustrated in Figure 1. This layer decouples file I/O from the compute nodes (*CN_i* in Fig 1), shipping those functions to the *forwarding nodes* instead, which are additional I/O nodes responsible for transferring I/O requests. It also enables compute nodes (1) to adopt a lightweight OS [48, 53, 64] that forwards file system calls to forwarding nodes, for higher and more consistent application performance [19], (2) to maintain fewer concurrent connections to the storage subsystem than having clients directly access file system servers, for better operational reliability, and (3) to facilitate the connection between two different network domains, typically set up with different topol-

[*]Most work conducted during appointment at Qatar Computing Research Institute.
[†]Wei Xue is the corresponding author. Email: xuewei@tsinghua.edu.cn

ogy and configurations, for computation and storage respectively. Finally, it provides an additional layer of prefetching/caching (or, more recently, burst buffer operations [51]), significantly improving user-perceived I/O performance and reducing backend data traffic.

Rank	Machine	Vendor	# C_node	# F_node	File system
3	Taihulight [11]	NRCPC	40,960	240	Lustre [24]
4	Tianhe-2A [69]	NUDT	16,000	256	Lustre + H2FS
5	Piz Daint [9]	Cray	6,751	54	Lustre + GPFS [57]
6	Trinity [17]	Cray	19,420	576	Lustre
9	Titan [15]	Cray	18,688	432	Lustre
10	Sequoia [10]	IBM	98,304	768	Lustre
12	Cori [2]	Cray	12,076	130	Lustre + GPFS
14	Oakforest-PACS [8]	Fujitsu	8,208	50	Lustre
18	K computer [5]	Fujitsu	82,944	5184	FEFS [56]

Table 1: I/O forwarding adopters among TOP20 machines (Nov 18)

Due to these advantages, I/O forwarding is quite popular, adopted by 9 out of the current TOP20 supercomputers (by the latest TOP500 list [16]). Table 1 summarizes their current TOP500 rankings and system configurations, including the number of compute and forwarding nodes. Note that recent Cray installations such as Cori and Trintiy use forwarding nodes with SSD-based burst buffers [3]. Forwarding architecture is also targeted in an Exascale storage design [45].

Despite the I/O forwarding layer's nature in decoupling compute nodes from backend storage nodes and enabling flexible I/O resource allocation, to provision a future system with forwarding resources (or to manage them for a current one) is challenging, as reasoned earlier. As a result, existing systems mostly adopt a *fixed forwarding-node mapping (FFM)* strategy between compute nodes and forwarding nodes, as illustrated in Figure 1. Though compute nodes are connected to all forwarding nodes, each forwarding node is assigned a fixed subset of k compute nodes to serve [48, 49, 63]. E.g., the compute-to-forwarding mapping is fixed at 512-1 at the No.3 supercomputer TaihuLight [30], and 380-1 at the No.5 Piz Daint [55].

This paper proposes a new method of forwarding resource provisioning. Rather than making fixed mapping decisions based on rough estimates, supercomputer owners could enable *dynamic forwarding resource allocation (DFRA)*, with flexible, application-aware compute-to-forwarding node mappings. We argue that DFRA not only alleviates center management's difficult hardware provisioning burdens, but significantly improves forwarding resource utilization *and* inter-application performance isolation.

DFRA is motivated by results of our whole-system I/O monitoring at a leading supercomputing center and extensive experiments. Specifically, we found the common practice of FFM problematic: (1) while the default allocation suffices on average in serving applications' I/O demands, the forwarding layer could easily become a performance bottleneck, leading to *poor application I/O performance and scalability* as well as *low backend resource utilization*; meanwhile the majority of forwarding nodes tend to stay under-utilized. (2) Forwarding nodes shared among relatively small jobs or partitions of large jobs become a contention point, where applications with conflicting I/O demands could inflict *severe performance interference* to each other. Section 2 provides a more detailed discussion of these issues.

Targeting these two major limitations of FFM, we devised a practical *forwarding-node scaling method*, which estimates the number of forwarding nodes needed by a certain job based on its I/O history records. We also performed an in-depth *inter-application interference study*, based on which we developed an interference detection mechanism to prevent contention-prone applications from sharing common forwarding nodes. Both approaches leverage automatic and online I/O subsystem monitoring and performance data analysis that require no user effort.

We implemented, evaluated, and deployed our proposed approach in the production environment of Sunway TaihuLight, currently the world's No.3 supercomputer. Deployment on such a large production system requires us to adopt practical and robust decision making and reduce software complexity when possible. In particular, we positioned DFRA as a "remapping" service, performed only when projected I/O time savings significantly offset the node-relinking overhead.

Since its deployment in Feb 2018, DFRA has been applied to ultra-scale I/O intensive applications on TaihuLight and has brought savings of bringing around 30 million core-hours per month, benefiting major users (who together consume over 97% of total core-hours). Our results show that our remapping can achieve up to 18.9× improvement to real, large-scale applications' I/O performance. Finally, though our development and evaluation are based on the TaihuLight supercomputer, the proposed dynamic forwarding resource allocation is not platform-specific and can be applied to other machines adopting I/O forwarding.

2 Background and Problems

Modern I/O forwarding architectures in HPC machines typically deploy a static mapping strategy [18] (referred to as FFM for the rest of the paper), with I/O requests from a compute node mapped to a fixed forwarding node. Here we demonstrate the problems associated with this approach, using the world's No.3 supercomputer TaihuLight as a sample platform. Specifically, we discuss *resource misallocation*, *inter-application interference*, and *forwarding node anomalies*, proceeded by introduction to the platform and the real-world applications to be discussed.

2.1 Overview of Platform and Applications

Platform Sunway TaihuLight is currently the world's No.3 supercomputer [30], with over 10M cores and 125-Petaflop peak performance. Its main storage system is a 10PB Lustre parallel file system [24], delivering 240GB/s and 220GB/s aggregating bandwidths for reads and writes respectively, using 288 storage nodes and 144 Sugon DS800 disk arrays. Between its compute nodes and the Lustre backend is a globally-shared layer of 240 I/O forwarding nodes. Each forwarding node provides a bandwidth of 2.5GB/s and plays a dual role, both as a Lightweight File System (LWFS) [6] server to the compute nodes and a client to the Lustre backend. Before our DFRA deployment, 80 forwarding nodes were used for daily service, the other 160 reserved as backup or for large production runs with whole-system reservations.

In addition, TaihuLight has an online, end-to-end I/O monitoring system, Beacon [21]. It provides rich profiling information such as average application I/O bandwidth, I/O time and I/O access mode, as well as real-time system load and performance measurements across different layers of TaihuLight's storage system.

Applications Our test programs include 11 real-world applications and one parallel I/O benchmark. Six of them are 2017 and 2018 ACM Gordon Bell Prize contenders: `CESM` [37] (Community Earth System Model) is an earth simulation software system which consists of many climate models; `CAM` [59] is a standalone global atmospheric model deriving from the `CESM` project for climate simulation/projection; `AWP` [25] is a widely-used earthquake simulator [26,54]; `Shentu` [35] is an extreme-scale graph engine; `LAMMPS` [68] (Large-scale Atomic/Molecular Massively Parallel Simulator) is a popular molecular dynamics software; `Macdrp` [23] is a new earthquake simulation tool, specializing in accurate replay of earthquake scenarios with complex surface topography. `CAM` and `AWP` were among the three 2017 Gordon Bell Prize finalists (`AWP` being the final winner), while `Shentu` is in the 2018 finalist.

Note that although all 6 applications above can scale to the full TaihuLight system's 40,000+ compute nodes, full-scale production runs are conducted mostly with pre-arranged system-wide reservation. In most cases, we do not have such reservation or the largest-scale input datasets to evaluate their maximum-scale executions. However, throughout the year, their developers and users conducted many mid-size runs, each using hundreds or thousands of compute nodes. Most of our experiments evaluate at such scale, where I/O performance improvement can save shared I/O resources and reduce application execution time. Meanwhile, our findings here remain applicable to larger-scale runs.

The remaining large-scale applications in our testbed are: `DNDC` [32] (biogeochemistry application for agroecosystems simulation), `WRF` [1] (regional numerical weather prediction system), `APT` [67] (particle dynamics simulation code), `XCFD` (computational fluid dynamics simulator), and `swDNN` [29] (deep neural network engine). For the ease of controlling I/O behaviors and execution parameters, we also use `MPI-IO` [7], a widely-used MPI-IO benchmark by LANL.

These programs represent diverse data access behaviors regarding request characteristics, I/O volume, I/O library, and file sharing mode. Table 2 summarizes their I/O profiles. Here we roughly label each application as "high" or "low" in three dimensions: I/O throughput, IOPS, and metadata operation intensity, using empirical thresholds.[1]

App	Throughput	IOPS	Metadata	I/O Lib	I/O Mode
$\texttt{MPI-IO}_N$	High	Low	Low	MPI-IO	N-N
`DNDC`	Low	Low	High	POSIX	N-N
`APT`	Low	High	Low	HDF5	N-N
$\texttt{WRF}_1$	Low	Low	Low	NetCDF	1-1
$\texttt{WRF}_N$	High	High	Low	NetCDF	N-N
`CAM`	Low	Low	Low	NetCDF	1-1
`AWP`	Low	Low	Low	MPI-IO	N-1
`Shentu`	High	High	Low	POSIX	N-N
`Macdrp`	High	Low	Low	POSIX	N-N
`LAMMPS`	High	Low	Low	MPI-IO	N-N
`XCFD`	High	Low	Low	POSIX	N-N
`CESM`	High	Low	Low	NetCDF	N-N
`swDNN`	Low	Low	Low	HDF5	N-N

Table 2: Summary of test programs' I/O characteristics. "N-N" mode means N processes operate N separate files. "N-1" means N processes operate on one shared file. "1-1" means only one process among all processes operates on one file.

2.2 Motivation 1: Resource Misallocation

As shown above, applications have drastically different I/O demands, some requiring a much lower compute-to-forwarding nodes ratio than others. Traditional FFM does not account for varying I/O behaviors across applications, leading to significant resource misallocation. Below we discuss concrete sample scenarios.

Forwarding node under-provisioning The default I/O forwarding node allocation of one per 512 compute nodes in TaihuLight is adequate for the majority of applications we have profiled, but severely low for the most I/O intensive applications, where the forwarding nodes become an I/O performance bottleneck. Due to the transparent nature of the forwarding layer, such bottleneck is often obscure and hard to detect by application developers or users.

Figure 2 demonstrates the impact of allocating more forwarding nodes to two representative real-world applications: `XCFD` and $\texttt{WRF}_1$. We plot the I/O performance speedup (normalized to that under the default allocation of one forwarding node), as a function of the number of exclusive forwarding nodes assigned to the application.

We find that `XCFD` benefits significantly from increased forwarding nodes. `XCFD` adopts an N-N parallel I/O mode, where each MPI process accesses its own files. Thus many backend storage nodes and OSTs (Object Storage Targets, Lustre term for a single exported backend object storage volume) are involved in each I/O phase, especially when N is large. In general, applications with such I/O behavior suffer under FFM, due to the limited processing bandwidth in the assigned forwarding nodes.

[1]Calculated by $\alpha\times$ per-forwarding-node peak performance. In this paper we set α as 0.4, resulting in thresholds of 1GB/s for throughput, 10,000 for IOPS, and 200/s for metadata operation rate, respectively.

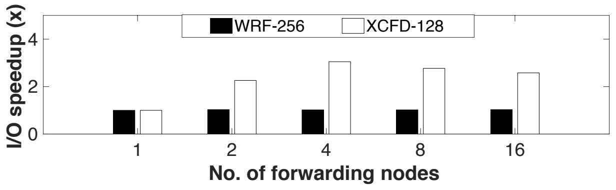


Figure 2: I/O performance speedup of WRF$_1$ and XCFD with increasing dedicated forwarding node allocation. For the rest of the paper, the number after application name gives the number of compute nodes used in execution.

Our I/O profiling on TaihuLight indicates that among jobs using at least 32 compute nodes, around 9% use the N-N I/O mode, potentially seeing significant performance improvement given more forwarding nodes. Such under-provisioning was observed on other supercomputers, e.g., recent Cray systems where I/O requests issued by a single compute node can saturate a forwarding node [27].

Applications like WRF$_1$, meanwhile, adopt the 1-1 I/O mode, where they aggregate reads/writes to a single file in each I/O phase. Intuitively, such applications do not benefit from higher forwarding node allocation. In addition, on TaihuLight applications with the 1-1 mode typically do not generate large I/O volumes in a single I/O phase, though they tend to run longer. Combining these two factors, 1-1 applications are mostly insensitive to additional forwarding layer resources beyond the default allocation.

Forwarding node load imbalance Application-oblivious forwarding resource allocation can lead to severe load imbalance across forwarding nodes. To verify this, we examined historical I/O traces collected on TaihuLight's forwarding nodes to check how they are occupied over time.

For every forwarding node, TaihuLight's profiling system records its per-second pass-through bandwidth. Analysis of such results first indicates that during the majority of profiled time intervals, the forwarding nodes are severely underutilized, echoing other studies' findings on overall low supercomputer I/O resource utilization [43, 47]. Meanwhile we

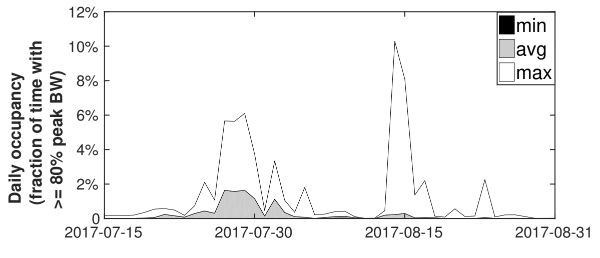


Figure 3: Sample TaihuLight forwarding layer load history

found high variability of loads across forwarding nodes and high day-to-day variances on forwarding node occupancy. We illustrate this with the forwarding nodes' *daily occupancy*, calculated as the fraction of 1-second windows in a day where a node's average bandwidth reaches 80% of the peak forwarding bandwidth of 2.5 GB/s. Figure 3 plots the minimum, average, and maximum daily occupancy across the 80 TaihuLight forwarding nodes, between July 15th and August 31st, 2017. We see both high variability in overall load (irregular average and maximum curves) and high load imbalance (large difference between the two).[2]

With recent and emerging systems adopting a burst buffer (BB) layer, such under-utilization and imbalance could bring wasted NVM spaces, buffer overflow, unnecessary data swapping, or imbalanced device wear.

2.3 Motivation 2: Inter-job Interference

I/O interference is a serious problem known to modern supercomputer users [28, 38, 40, 44, 61]. The common FFM practice not only neglects individual applications' I/O demands, but also creates an additional contention point by sharing forwarding nodes among concurrent jobs with conflicting I/O patterns. Figure 4 illustrates this using three real applications: AWP Shentu, and LAMMPS.All used the default 512-1 compute-to-forwarding mapping.

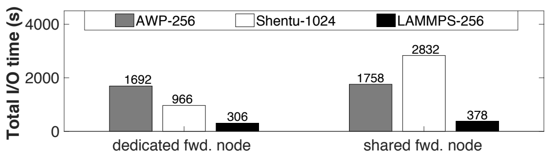


Figure 4: I/O performance impact of forwarding node sharing

We tested two execution modes, with each application allocated *dedicated* forwarding nodes vs. applications using *shared* ones. In both modes all three applications ran simultaneously. Note that for Shentu in the shared mode, it was allocated one dedicated forwarding node and two more nodes to share with other applications: one with AWP and one with LAMMPS, which were each running on 256 compute nodes (and thus allocated *half* of a forwarding node each).

As expected, all three experienced faster I/O with dedicated forwarding nodes. However, some suffered much higher performance interference. While AWP and LAMMPS saw mild slowdowns (4% and 23% increase in total I/O time), Shentu had a 3× increase. This is due to the highly disruptive behavior of AWP's N-1 I/O mode (discussed in more details later), causing severe slowdown of Shentu processes accessing the same forwarding node. Given the synchronous nature of many parallel programs, their barrier-style parallel I/O operations wait for all processes involved to finish. Thus slowdown from the "problem forwarding node" shared with AWP is propagated to the entire application, despite that it had one dedicated forwarding node and shared the final one with a much more friendly LAMMPS.

In Section 5, we present an in-depth inter-application interference study, based on which we perform application-aware interference estimation to avoid sharing forwarding nodes among applications prone to interference.

[2]Our recent paper on TaihuLight's Beacon monitoring system gives more details on workload characteristics [21].

2.4 Motivation 3: Forwarding Node Anomaly

Finally, when certain forwarding nodes show abnormal behavior due to software or hardware faults, applications assigned to work through these *slow* nodes under FFM would suffer. We found TaihuLight forwarding nodes prone to correctable failures in memory or network, confirming the "fail-slow" phenomenon observed at data centers [33].

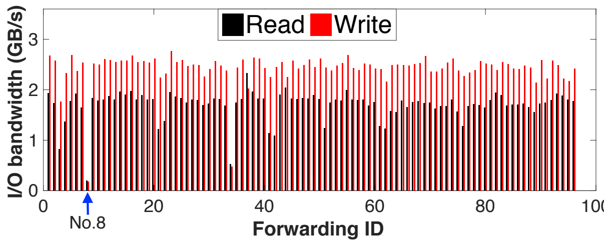


Figure 5: Forwarding node peak performance. Forwarding nodes with IDs 3, 8 and 34 show abnormal performance.

Figure 5 shows sample benchmarking results measuring read/write bandwidth across 96 currently active forwarding nodes, conducted during system maintenance. While most forwarding nodes do report consistent bandwidth levels (with expected variability due to network contention, disk status, etc.), a small number of them clearly exhibit performance anomalies. In particular, forwarding node No.8 (highlighted with arrow) is an obvious outlier, with average read and write bandwidth at 7% and 12% of peak, respectively.

Fortunately, the I/O monitoring system in TaihuLight performs routine, automatic node anomaly detection across all layers of the I/O infrastructure. As shown in Section 3, our proposed dynamic forwarding system leverages such anomaly detection to skip nodes experiencing anomalous behavior in its dynamic allocation.

3 System Overview

Given the above multi-faceted problems caused by FFM, we propose a practical-use and efficient dynamic forwarding resource allocation mechanism, DFRA. DFRA works by *remapping* a group of compute nodes (scheduled to soon start executing an application) to other than their default forwarding node assignments, whenever the remapping is expected to produce significant application I/O time savings. It serves three specific purposes: (1) to perform application-aware forwarding node allocation to avoid resource under-provisioning for I/O-intensive jobs, (2) to mitigate inter-application performance interference at the forwarding layer, and (3) to (temporarily) exclude forwarding nodes identified as having performance anomalies.

To remap at the job granularity does not pose much technical difficulty by itself. The challenge lies in developing an automatic workflow that examines both the application's I/O demands and real-time system status, and performs effective inter-application I/O interference estimation, while remaining as transparent as possible to users and relieving administrators from labor-intensive manual optimizations.

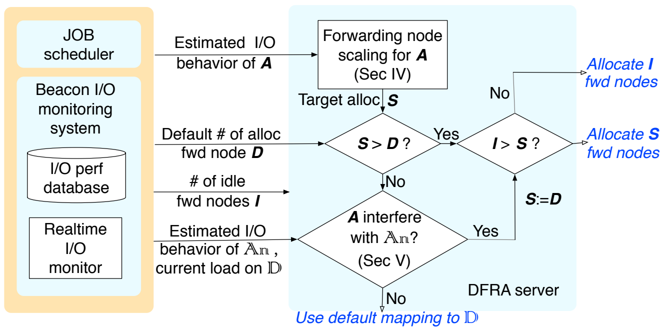

JOB scheduler
Beacon I/O monitoring system
I/O perf database
Realtime I/O monitor
Estimated I/O behavior of A
Default # of alloc fwd node D
# of idle fwd nodes I
Estimated I/O behavior of $\mathbb{A}_n$, current load on $\mathbb{D}$
Forwarding node scaling for A (Sec IV)
Target alloc, S
S > D ?
Yes
No
I > S ?
Allocate I fwd nodes
Allocate S fwd nodes
S:=D
A interfere with $\mathbb{A}_n$? (Sec V)
Yes
No
DFRA server
Use default mapping to $\mathbb{D}$

Figure 6: DFRA decision-making workflow

To this end, we leverage the Beacon I/O monitoring system on TaihuLight to perform continuous application I/O profiling and learn the I/O characteristics of applications from history.[3] Assisted with all-layer profiling data, from the compute nodes to the backend OSTs, *plus* per-job scheduling history that reveals the mapping of a job's processes to compute nodes, we obtain a detailed understanding of each past job's I/O behavior, including peak bandwidth per compute node, request type/size distribution, periodic I/O frequency, I/O mode (N-N, N-1, 1-1, etc.), and metadata access intensity. Given that HPC platforms typically see applications run repeatedly, with very similar I/O patterns [62], there is high likelihood that the past reflects the future.

We have designed, implemented, and deployed a dynamic forwarding resource allocation mechanism on TaihuLight, as depicted in Figure 6 It determines whether a *target job* A, scheduled to begin execution on a certain set of compute nodes, needs to have forwarding nodes remapped and if so, to which nodes. Implementation-wise, such proposed dynamic forwarding resource allocation component resides on a single dedicated server (DFRA server). It interacts with the Beacon I/O monitoring system and the job scheduler. Beacon provides an I/O performance database to query using A's job information (e.g., application name, project name, user name, and execution scale) and estimates its I/O characteristics based on historical records.

The job's expected I/O features, such as I/O mode and the number of compute nodes performing I/O, are then fed to the DFRA server. First it checks whether this application needs to *scale out* to use more forwarding nodes. If not (the more likely case), then we incorporate real-time scheduling information to know about the "neighbor applications" $\mathbb{A}_n$ (the set of applications currently running on $\mathbb{D}$, the forwarding nodes to be assigned under the default allocation). This allows the DFRA server to check whether the default mapping will produce significant *performance interference* with neighbors already running there. If significant interference is expected, we keep the default allocation size D, but would

[3]Partial I/O traces released at https://github.com/Beaconsys/Beacon

remap the compute nodes to dedicated forwarding nodes.[4]

If scaling *is* required, we first calculate S, the number of forwarding nodes needed. We then check I, the number of *idle* forwarding nodes currently available, excluding those undergoing performance anomaly, and allocate the fewer between S and I. Though more sophisticated "partial-node" allocation is possible, we choose the more simple scheme considering the overall forwarding node under-utilization.

In summary, there are two types of "upgrades": to grant more forwarding nodes (for capacity) or grant unused forwarding nodes (for isolation). In both cases, as we only allocate *dedicated nodes* from the idle pool, no interference check is further needed. In the specific case of TaihuLight, at the beginning of this research, beside the 80 forwarding nodes using the default 512-1 mapping, more than 100 are reserved for backup or manual allocation. For systems without such over-provisioning, we recommend the default allocation be lowered to serve the majority of jobs, who are not I/O-intensive, and have a set of "spare" forwarding nodes for ad-hoc remapping.

Note that this is a best-effort system transparent to users. Additionally, for the majority of applications, who are not I/O-intensive enough to warrant higher allocation and not significantly interference-prone with expected neighbors, the decision is to remain with default mapping.

The actual remapping process is conducted upon the jobs' dispatch and involves making RPCs from the DFRA server to the compute nodes concerned, instructing them to drop off the original connection and connect to newly assigned forwarding nodes, allocated from the current available forwarding node pool. Considering that a job tends to have consistent I/O behavior, this remapping is done once per job execution, rather than per request. If remapped, when A completes, its compute nodes will be reset to default mapping, making DFRA maintenance simple and robust.

4 Automatic Forwarding Node Scaling

To decide on the "upgrade eligibility" of a job, we estimate its multiple I/O behavior metrics based on the query results of I/O monitoring database. When historical information is not sufficient, e.g., as in the case of new applications, our system does *not* change the default mapping. I/O monitoring data collected from these runs will help forwarding resource allocation in future executions.

Our scaling decision-making adopts a per-job forwarding node allocation algorithm. It considers both the application-specific I/O workload characteristics and historical performance data of forwarding node load levels while serving this application. Most of the threshold values are set empirically according to our extensive benchmarking of the system, and can be further adjusted based on continuous I/O performance monitoring. More specifically, the target job A needs to meet the following criteria to be *eligible for a higher forwarding resource allocation* than the default setting:

1. its total I/O volume is over V_{min} during its previous execution;
2. it has at least N_{min} compute nodes performing I/O; and
3. it is *not* considered metadata-operation-bound, i.e., its past average number of metadata operations waiting at a forwarding node's queues is under $W_{metadata}$.

The rationale is based on the primary reason for a job to have an upgraded allocation: it possesses enough I/O parallelism to benefit from more forwarding resources. For such benefit to offset the forwarding node remapping overhead, first the application needs to generate a minimum amount of I/O traffic. Applications diagnosed as metadata-operation-heavy, regardless of their total I/O volume or I/O parallelism, are found to not benefit from more forwarding nodes as their bottleneck is the metadata server (MDS).

If A passes this test, with past history showing that it is expected to use N_A of its compute nodes to perform I/O, the number of its forwarding node allocation S is calculated as $\lceil N_A/F \rceil$. Here F is a *scaling factor* that reflects typically how many I/O-intensive compute nodes can be handled by a forwarding node without reaching its performance cap. In our implementation, F is set as $\lceil B_f/B_c \rceil$, where B_f and B_c are the peak I/O bandwidths of a single forwarding and compute node, respectively. If not enough idle forwarding nodes are available, we allocate all the available nodes. We expect this case to be extremely rare, as given the typical system load, there are enough idle forwarding nodes to satisfy all allocation upgrades.

In our deployment on TaihuLight, we empirically set V_{min} at 20 GB, N_{min} at the F value (32 based on the formula above), and $W_{metadata}$ at twice the per-forwarding-node thread pool size, also 32. These can be easily adjusted based on machine specifications and desired aggressiveness.

We do *not* downgrade allocations for the metadata-heavy or 1-1 I/O mode jobs, considering their baseline per-process I/O activities (such as executable loading, logging, and standard output). Also considering TaihuLight's sufficient backup forwarding nodes, we opt not to pay the remapping overhead for downgrading allocations in this deployment, though downgrading is easy to implement when needed.

Figure 7 shows how application I/O performance, in aggregate I/O bandwidth measured from the application side, changes with different compute-to-forwarding node ratios. As these tests used dedicated forwarding nodes, we started from the 256-1 allocation, rather than the default 512-1.

Here several applications, namely `APT`, `DNDC`, `WRF`$_1$, and `CAM`, due to insufficient I/O parallelism or being metadata-heavy, do not pass the eligibility test. Their I/O performance results confirm that they would have received very little performance improvement with more forwarding nodes been allocated. The other applications, however, see substantial I/O

[4]This does not consider the jobs' duration, as history records or job script specified run times are not reliable indicators. Such conservative strategy is allowed by the typical abundance of idle forwarding nodes.

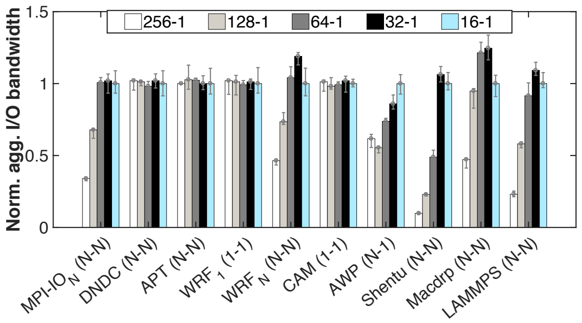


Figure 7: Aggregate I/O bandwidth under increasing forwarding node allocation (compute-to-forwarding mapping ratio), normalized to the 16-1 case. All applications ran using 256 compute nodes (1024 MPI processes). Each test was repeated 5 times, with average results plotted and error bars giving 95% confidence intervals.

bandwidth enhancement with increased forwarding node allocations, by up to a factor of 10.9×. Judging from results across all such applications, our current F setting of 32 deliver best aggregate I/O bandwidth in most cases.

5 Interference Analysis and Avoidance

Our DFRA system attempts to mitigate this performance interference by assigning jobs that are expected to interfere to *different* forwarding nodes. Note that prior work on interference detection and optimization focused mostly on deriving offline, analytical interference models (e.g., [28,31]). In contrast, our work focuses on designing practical online interference estimation techniques that DFRA can use effectively.

5.1 Inter-application Interference Analysis

We first conduct a rather controlled study, to observe I/O interference behavior between pairs of representative I/O applications. From the applications described in Table 2, we select eight that cover different I/O resource-consumption patterns. Next, we perform pairwise co-execution among these selected applications. For this, we use 256 compute nodes (1024 MPI processes) each, so that the paired workloads have equal execution scale. Under the default allocation, the 512 compute nodes running the two programs hence share one forwarding node.

To gauge interference, we measure each application's *I/O slowdown* by calculating the application's relative slowdown factor in overall I/O performance (the time spent in the I/O interference interval) from that of its solo run. Table 3 shows the pairwise results, with high-interference pairs (with either slowdown factor >3) marked in **bold**, and medium-interference ones (those among the rest with either slowdown factor >2) marked with "*".

The majority of our applications in this study are intensive in at least one dimension of I/O resource usage and are expected to see I/O performance slowdown when they share the same I/O path. Results in Table 3 confirm this. An application exhibits an I/O slowdown of around 2× when co-running with itself (another instance of the same application), due to the expected resource contention. The remaining pairwise slowdown results reveal several interesting interference behaviors.

First, we find that applications with *low* demands in all three dimensions (throughput, IOPS, and metadata operation rate) do not introduce or suffer significant I/O slowdown when co-running with other applications, with the exception of applications using the N-1 I/O mode (recall Table 2).

To understand the reasons behind, we conducted follow-up investigations. The three applications that fall into the "Low/Low/Low" category are $\texttt{WRF}_1$, `CAM`, and `AWP`. Among them, `AWP` turns out to be a highly disruptive workload, causing high degrees of I/O slowdown to whoever runs with it. We performed additional experiments, including `MPI-IO` tests emulating its behavior with different I/O parameters, and identified the problem being its N-1 file sharing mode. While N-1 writes have been notoriously slow (such as with Lustre [20], also verified by our own benchmarking), our study reveals that it brings high disturbance (average of 38.4× to other applications tested).

Further examination of profiling results identified the forwarding layer as the source of interference. Each forwarding node maintains a fixed thread pool, processing client requests from the compute nodes it is in charge of. While designed to allow parallel handling of concurrent client requests, applications using the N-1 file sharing mode generate a large number of requests and flood the thread pool. Their occupation of the forwarding layer thread resources is further prolonged by the slow Lustre backend processing of such I/O requests (often involving synchronization via locks). The result is that other concurrent applications, whose I/O requests might be far fewer and more efficient, are blocked waiting for thread resources, while the I/O system remains under-utilized.

Such effect is highlighted by follow-up test results in Figure 8. We pair 2 benchmarks, $\texttt{MPI-IO}_1$ (N-1) and $\texttt{MPI-IO}_N$ (N-N), running at different scales. The bars (left y axis) show the queue lengths of pending requests at the forwarding layer. While the queue length increases proportionally to the number of compute processes, as expected, the "co-run" queue length of $\texttt{MPI-IO}_N$ does not grow significantly from its solo run. The much greater increase in $\texttt{MPI-IO}_N$ latency (red curves using the right y axis), meanwhile, comes from the slowdown of each $\texttt{MPI-IO}_1$ request.

Secondly, we observe from Table 3 that `DNDC` introduced significant slowdown to all other workloads (by a factor from 2.4× to 33.3×). A closer look finds that `DNDC` is the only application in our testbed with significant metadata access intensity. `DNDC`'s production runs are not particularly large (only using 2048 processes), which simultaneously read 64,000 small files (up to several KBs each). The large number of open/close requests pile up and block requests from other applications obviously.

More profiling reveals that read requests see much faster

Apps	MPI-IO$_N$	APT	DNDC	WRF$_1$	WRF$_N$	Shentu	CAM	AWP
MPI-IO$_N$	*(2.1, 2.1)	**(1.1, 9.3)**	**(4.8, 1.1)**	(1.0, 1.0)	*(2.1, 2.0)	**(1.3, 4.5)**	(1.0, 1.0)	**(3.3, 1.1)**
APT	-	*(2.0, 2.1)	**(33.3, 1.0)**	(1.0, 1.0)	**(4.3, 1.4)**	**(6.3, 1.3)**	(1.0, 1.0)	**(50.0, 1.1)**
DNDC	-	-	*(2.0, 2.0)	**(1.0, 25.0)**	**(1.0, 11.1)**	**(1.1, 16.7)**	**(1.0, 33.3)**	*(2.2, 2.4)
WRF$_1$	-	-	-	(1.0, 1.0)	(1.0, 1.0)	(1.0, 1.0)	(1.0, 1.0)	**(50.0, 1.0)**
WRF$_N$	-	-	-	-	*(2.1, 2.1)	*(2.0, 2.3)	(1.0, 1.0)	**(12.5, 1.3)**
Shentu	-	-	-	-	-	*(2.0, 2.0)	(1.0, 1.0)	**(12.5, 1.1)**
CAM	-	-	-	-	-	-	(1.0, 1.0)	**(100.0, 1.0)**
AWP	-	-	-	-	-	-	-	*(2.0, 2.0)

Table 3: I/O slowdown factor pairs of applications listed in row and column headers. *E.g.*, in the 1st row, 2nd column, MPI-IO$_N$ has slowdown of 1.1 and APT has 9.3 when they co-execute. (**Bold** and "*" indicate high- and medium-interference, respectively)

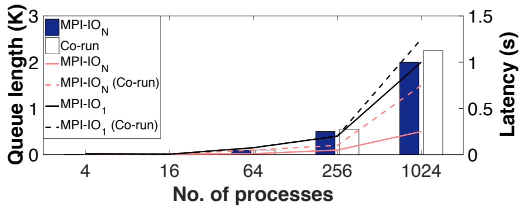


Figure 8: Interference between MPI-IO$_1$ and MPI-IO$_N$. Bar represents queue length and line represents latency.

processing than open, with only slight increase in processing time when DNDC joins a read-heavy job, indicating bottleneck-free Lustre handling. The wait time, however, sees almost 4× increase for read and around 2× for open operations. Besides that the forwarding node thread pool being the point of contention, the asymmetric delay prompted us to examine its scheduling policy. We found that metadata requests were given higher priority over normal file I/O, favoring interactive file system user experience. This, combined with their longer processing time, makes metadata-heavy applications like DNDC unsuspected disruptive workloads. While our ongoing work targets more adaptive policies, for DFRA we specifically check jobs' metadata operation intensity for interference estimate.

Finally, we find that even applications with seemingly *orthogonal* resource usage patterns may not get along well, with *asymmetric* performance impact on each other. In particular, we find that high-bandwidth, low-IOPS applications impact the performance of low-bandwidth, high-IOPS ones (but not vice versa). This can be seen from the APT-MPI-IO$_N$ results in Table 3, with the high-IOPS APT suffering an almost 10× slowdown while the high-bandwidth MPI-IO$_N$ is hardly impacted. A closer look reveals that APT reaches IOPS of over 80,000, with requests sized under 1KB. The reason behind the asymmetric slowdown is then intuitive: high-bandwidth applications likely perform sequential I/O with large request sizes, which force the many small requests from the high-IOPS applications to wait long.

In summary, we discover that I/O interference not only comes from bandwidth-intensive applications and problematic access patterns (as assumed by previous studies [31,44]), but also from applications issuing inefficient I/O requests, while *simultaneously incurring high contention and low utilization*, such as in the metadata-heavy and high-IOPS cases.

5.2 Inter-job Interference Estimate for DFRA

We now discuss DFRA's inter-application interference check, introduced in Section 5.1. Recall that it is needed only when we decide that the target job A, which is to be scheduled, does not need more forwarding nodes than granted by the default mapping. The interference check is then performed pairwise, between A and each member of its neighbor application set $\mathbb{A}_n$.

As actual I/O interferences incurred during co-executions of applications depend on other factors such as their I/O phases' frequency and interleaving, we use our interference analysis results to make conservative, qualitative decisions. More specifically, for A and each of its neighbor in $\mathbb{A}_n$, we consider interference is likely if either A or the neighbor is:

1. using the N-1 I/O mode, *or*
2. considered "metadata operation heavy" (average number of metadata operations waiting at a forwarding nodes queue $> W_{metadata}$), *or*
3. considered "high-bandwidth" or "high-IOPS" (using criteria described in Section 2.1).

For A, the above check has to be based on our monitoring system's per-application I/O performance history data. For jobs in $\mathbb{A}_n$, however, our history-based I/O behavior inference can and should be complemented with real-time I/O behavior analysis. In particular, as the inferred I/O behavior includes pattern information such as I/O phase frequency, I/O volume per process performing I/O, and I/O mode, such estimates can be verified by actual data collected during the neighbors' current execution. E.g., if a forwarding node is receiving unexpectedly low I/O load from an application running, DFRA considers the application turns off I/O for this run, overriding its positive interference estimate. Similarly, if an application is issuing I/O at intensity not indicated by its past history, we play safe and use the peak load level measured during its execution so far on the forwarding node(s) involved, to determine whether interference is likely.

6 Evaluation

6.1 Job Statistics from I/O History Analysis

First, DFRA's working relies on applications' overall consistency in I/O behavior. We verified this with the 18-month TaihuLight I/O profiling results, confirming observations by existing studies [31, 44]. Specifically, if we simply forecast a new job's I/O mode and volume as those in its latest run using the same number of compute nodes, we can successfully predict these parameters with under 20% deviation for 96,621 jobs (90.3%) out of 107,001 in total.

Category	Count	Count(%)	Core-hour(%)
Total jobs	107,001	100%	100%
Job benefits from DFRA	14,712	13.7%	79.0%
Job's I/O volume $< V_{min}$ (20 GB)	83,562	78.1%	18.9%
Job's I/O nodes $< N_{min}$ (32)	8,727	8.2%	2.1%
Job's metadata queue length $\geqslant W_{metadata}$ (32)	0	0.0%	0.0%

Table 4: DFRA eligibility screening results, based on using per-job I/O history between April 2017 and August 2018

We then give statistics about DFRA's decisions and its potential beneficiaries, by running these 18-month job I/O profiles through DFRA's scaling decision making. For jobs that were refused allocation upgrades, we categorize them by the first test failed during the DFRA allocation scaling eligibility check (Section 4). Table 4 lists the results.

First, 13.7% jobs (minority in count yet accounting for 79.0% of core-hours) are granted upgrades and expected to benefit from DFRA. This demonstrates that though the I/O system is overall underutilized, there are substantial amount of I/O-intensive jobs as potential beneficiaries. Among the rest, most fail to meet the total I/O volume threshold V_{min}, followed by the number of I/O nodes involved. No job fails at the metadata-intensity check, as such applications in this particular job history do not pass the I/O volume test.

Also, throughout this history "replay" using DFRA, the average forwarding node consumption is 171.2, suggesting that DFRA can get much better I/O performance while working well under the total 240-node forwarding capacity.

6.2 Performance/Consistency Improvement

Next we examine the impact of DFRA's deployment on real applications' I/O performance in the TaihuLight production environment. We run the 11 applications (introduced in Section 2.1) each for 10 times at randomly selected times during a 1-month period, each time under DFRA and FFM within the same job execution, with remapping done in between. To control total resource usage, `Shentu`, `LAMMPS`, and `Macdrp` run with 1,024 compute nodes, with the other applications run at their typical mid-size run scale (`swDNN` using 512 nodes while the rest using 256). They are further divided into two groups: *scaling*, with more forwarding nodes granted by DFRA, and *non-scaling*, with dedicated forwarding node allocation if deemed interference-prone by DFRA (which may depend on their neighbor jobs under the default mapping, though `APT` and `DNDC` are always isolated).

DFRA brings an average I/O speedup of 3.5× across all 11 applications, from 1.03×(`CAM`) to 18.9×(`Shentu`). As expected, applications in the scaling group receive higher speedup (average at 4.8× and up to 18.9×), while non-scaling applications benefit more from reduced performance variability (and potential slowdown incurred on their neighbors). However, the scaling group also obtains dramatic improvement in *I/O performance consistency*, with average reduction of 91.1% in range of I/O times.

The reason lies in the "mis-alignment" of compute nodes to forwarding nodes using FFM. Our job history finds over 99% of large-scale jobs (using 512 compute nodes or more) assigned to share forwarding nodes with other jobs, though their job scales are often perfect multiples of the default factor of 512. Intuitively, such fragmentation often also leads to dramatic load imbalance across forwarding nodes (partially) serving the same I/O-intensive application.

App	Comp. time	I/O time w. FFM	I/O time w. DFRA	Total time reduction
`Shentu`-1024	1,303s	1,204s	64s	45%
`LAMMPS`-1024	3,510s	431s	97s	8%
`Macdrp`-1024	6,932s	260s	105s	2%
`swDNN`-512	0s	132,710s	31,476s	76%
`AWP`-256	2,301s	255s	204s	2%
`CESM`-256	4,742s	942s	846s	2%
$\texttt{WRF}_N$-256	1,640s	135s	89s	3%
`DNDC`-256	992s	222s	216s	0.5%
$\texttt{WRF}_1$-256	1,640s	513s	479s	2%
`APT`-256	222s	46s	24s	8%
`CAM`-256	3,226s	899s	876s	0.6%

Table 5: Per-phase computation and I/O time of applications

Table 5 describes the impact of DFRA on resource-intensive applications' overall performance. All applications but one (`swDNN`) have clear repeated phases alternating between computation and I/O, while the number of such computation-I/O cycles may vary across runs according to users' needs. Therefore we illustrate the relative impact by listing the more stable per-cycle average computation time, average I/O time (with FFM and DFRA respectively), and the percentage of total time saving by DFRA. The last column does not change when a particular production run adjusts the number of computation-I/O cycles. `swDNN`, unlike timestep numerical simulations, is a parallel deep learning model training application that has fine-grained, interleaving computation and I/O, therefore we treat its execution as a single I/O phase.

As most applications conform to their "total I/O budget" by adjusting their I/O frequency, by taking one snapshot every k computation timesteps, DFRA is not expected to yield significant overall runtime reduction, especially with the non-scaling ones. However, it does bring impressive total time savings for I/O-bound applications `Shentu` (45%) and `swDNN` (76%), as well as over 8% savings for `APT` and `LAMMPS`. Meanwhile, making I/O faster also implies that the applications could afford to output more frequently under the same I/O budget.

Figure 10 illustrates this scenario using a `Shentu` test run using 1024 compute nodes, which are not allocated contiguously. Under DFRA, its 32 dedicated forwarding nodes serve equal partitions of compute nodes, as each compute node can be individually remapped to any forwarding node, allowing almost all compute nodes finishing I/O simultaneously. Under FFM, instead, these dispersed 1024 compute nodes are

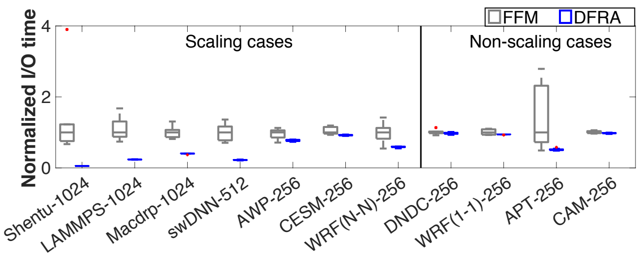


Figure 9: Impact of DFRA on application I/O performance, with results normalized to median I/O time under FFM

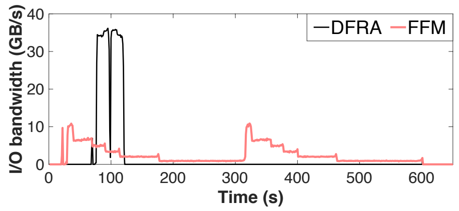
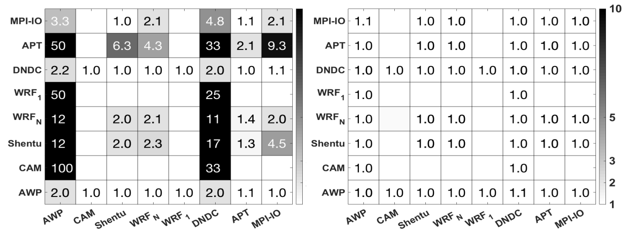


Figure 10: Sample `Shentu`-1024 I/O bandwidth timelines

mapped to 7 forwarding nodes. As a result, the same I/O activities take much longer, with multiple stair-steps produced by completion of different forwarding nodes.



(a) Before DFRA (b) After DFRA

Figure 11: Impact of DFRA's interference avoidance on pairwise application co-run slowdown. Darkness of each block reflects the slowdown factor of the application at row header by the application at the column header. Blocks with slowdown factor values give co-running pairs with interference anticipated by DFRA and hence allocated separate forwarding resources. In all experiments, the compute-to-forwarding mapping uses the default setting (512-1), with DFRA allocating dedicated forwarding nodes to application pairs it considers interference-prone.

To evaluate our proposed interference avoidance, we rerun the pairwise experiments (see Section 5) with DFRA on TaihuLight. Results are in Figure 11. We found DFRA can detect potential interference with pairs having slowdown factors over 1.1 at either side. In this test, we only separate these applications, without scaling up forwarding nodes, to isolate the benefits brought by interference avoidance.

Compared with the left plot, where just by sharing a forwarding node, certain applications could perceive a 2× to 100× I/O slowdown, the right plot reduces such slowdown to uniformly under 1.1×. With many jobs on TaihuLight sharing forwarding nodes, DFRA removes the infrequent (yet highly damaging) inter-application interference cases.

Finally, we evaluate an alternative approach, *RR*, which maps compute nodes to forwarding nodes in a round-robin manner. We test RR 32-1, where each group of contiguous 32 compute nodes are assigned to one forwarding node. Figure 12 gives the speedup (again over the 512-1 fixed allocation) of running one of the 5 applications given at the x-axis simultaneously with either `DNDC` or `AWP`. Each application runs on 256 compute nodes, with two co-running applications sharing 8 forwarding nodes using RR. For fair comparison, DFRA uses 64-1 allocation here, so that all co-run experiments enlist 8 forwarding nodes in total. RR spreads the load of each application to all 8 forwarding nodes, but does not offer the performance isolation brought by DFRA, when two applications running on disjoint compute nodes get mapped to common forwarding nodes. DFRA gives the two applications each a 64-1 *dedicated* allocation, delivering much higher I/O speedup in most cases, *plus* performance isolation from co-executing applications.

6.3 DFRA Decision Analysis

We now validate DFRA's forwarding node scaling decisions. Figure 13 shows, *in log scale,* performance of `MPI-IO` benchmarks with parameters uniformly sampled from a range, to adopt different I/O modes (N-1, N-N and N-M), I/O performing nodes, I/O request sizes, and metadata operation ratios. All tests are again divided into the *scaling* and *non-scaling* groups, referring to cases where the DFRA automatic scaling decision making processes chose to upgrade a job's forwarding node allocation, or retain the default one. The final results for both cases are consistent with the estimations projected by DFRA. Scaling cases can achieve on average 2.6× speedup (min at 1.1× and max at 7.1×), while the non-scaling ones' performance receives only trivial performance improvement (up to 1.05×).

Next we further examine the effectiveness of DFRA scaling, by measuring the queue length and I/O bandwidth of real-world applications on TaihuLight. Figure 14 shows results, again in log scale, with representative applications covering all I/O categories mentioned in Table 2. Among them, `DNDC` and `APT` are "non-scaling": `DNDC` is metadata-intensive and `APT` issues a large number of small-size I/O requests. We find their performance bottleneck not at the forwarding layer, explaining their little improvement in queue length and bandwidth when given more forwarding nodes. `AWP` adopts an N-1 I/O mode, generating high request pressure for forward-

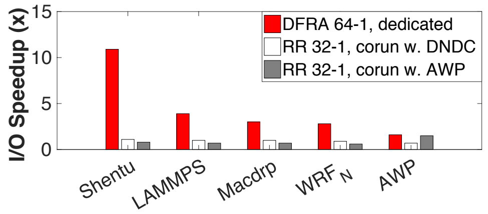


Figure 12: Speedup over 512-1 fixed allocation baseline, with two applications co-running, each using 256 compute nodes. Note that with its dedicated allocation, DFRA's performance is not impacted by co-running applications.

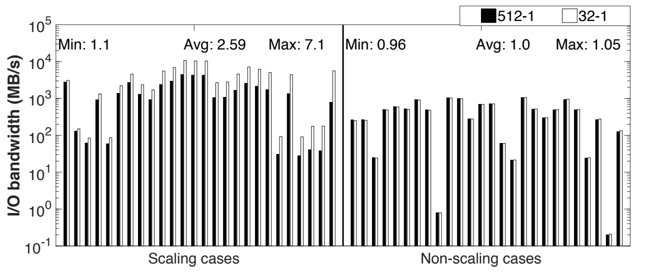
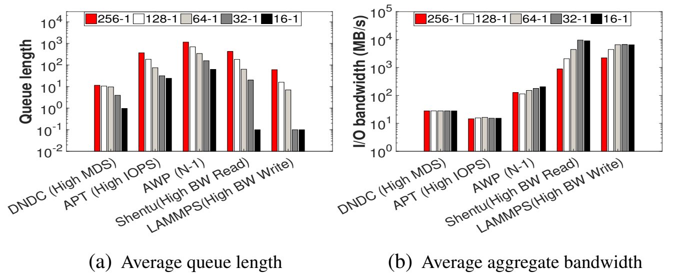


Figure 13: Scaling decision validation using `MPI-IO` benchmark instances. All experiments fix the total number of compute nodes to 256, with different number of them performing I/O, and run with 2 different compute-to-forwarding mapping ratios: 512-1 and 32-1. Results are sorted by speedup. Above the bars we list the minimum, average, and maximum speedup for each group.



(a) Average queue length (b) Average aggregate bandwidth

Figure 14: Average queue length (average number of requests pending in the queue, sampled at 0.01-second intervals) and I/O bandwidth with varying compute-to-forwarding mapping ratios during I/O execution. All applications use 256 compute node (1024 processes), with dedicated forwarding nodes.

ing nodes, thus receiving significant queue length improvement. Both `Shentu` and `LAMMPS` are bandwidth-hungry, benefiting significantly from the bandwidth side. In particular, `Shentu` gets a higher speedup as scaled-up allocation soothes its forwarding-side cache trashing.

6.4 Node Anomaly Screening

DFRA could screen out the abnormal forwarding nodes automatically. During our investigation, anomaly on forwarding nodes occurs for 6 times from Apr 2017 to Aug 2018. Jobs using such abnormal forwarding nodes typically experience substantial performance degradation. Figure 15 shows the performance impact when jobs get allocated an abnormal forwarding node. The I/O performance could see a 20× slowdown, due to the explicit barriers common with parallel

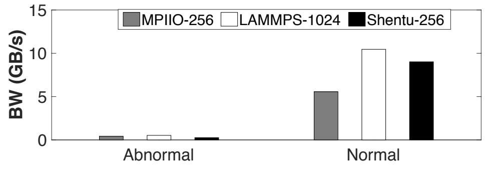


Figure 15: Performance impact when jobs run on abnormal forwarding nodes. All applications run with computing-forwarding ratio of 32-1.

I/O, forcing all processes to wait for the slow progress of the impaired forwarding node.

6.5 Overhead and Overall Resource Saving

Here we assess DFRA's overhead in performing the actual node remapping, while the allocation decision itself takes under 0.1s in all tests on TaihuLight. Figure 16 shows the average remapping time cost for different job sizes, plus the corresponding job dispatch time (without remapping) for reference. Though the remapping overhead increases linearly when more compute nodes are involved, it composes a minor addition to the baseline job dispatch overhead (the latter mainly due to compute nodes' slow wake-up from their power saving mode).

Note that this overhead is offset by our conservative screening based on jobs' past I/O profile. Even with 16,384 compute nodes, such minor delay in job dispatch is negligible compared with the total time saved in I/O phases, especially for long-running jobs. Since its deployment in Feb 2018, DFRA has brought an average execution time saving of over 6 minutes (up to several hours) to I/O-intensive jobs eligible for its remapping, estimated by comparing the I/O bandwidth benchmarked with the same application at the same job scale, before and after DFRA. Going over the actual TaihuLight job history, we thus estimate DFRA's overall resource saving at over 200 million of core-hours.

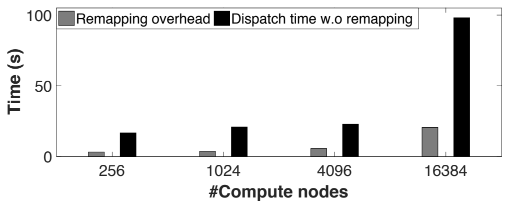


Figure 16: Average dynamic forwarding node remapping overhead

6.6 Extension to Burst Buffer Allocation

Finally, we briefly report our recent effort to apply DFRA techniques to dynamic allocation of burst buffer (BB) resources. We setup a testbed following the BB construction adopted by a previous study [41], containing 8 forwarding nodes, each with one 1.2TB Memblaze SSD to compose remote shared burst buffers.

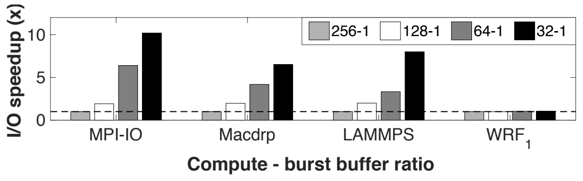


Figure 17: I/O speedup with different compute-to-BB node ratio, over performance with a baseline 256-1 allocation. Results are average from 3 tests, with error bars omitted due to small variance.

Figure 17 shows the performance impact of scaling up BB node allocations. All runs use 256 compute nodes. Not surprisingly, the more I/O-intensive applications (using N-N or

N-1) benefit significantly from more BB nodes, while the 1-1 mode $\mathtt{WRF}_1$ sees little improvement. The similarity between such result and that with forwarding resource scaling suggests that DFRA is promising for BB layer management as well. To this end, the next generation Sunway supercomputer will adopt DFRA, including for its planned BB layer.

7 Related Work

I/O forwarding design and optimization Cplant [12] first introduces the I/O forwarding layer, but without support for data caching or request aggregation. I/O forwarding then became popular at extreme scale in IBM Blue Gene (BG) platforms [14, 36, 48]. IOFSL [13] is an open-source, portable, high performance I/O forwarding solution that provides a POSIX-like view of a forwarded file system to an application. The Cray XC series uses Data Virtualization Service (DVS) [4] for I/O forwarding. Our proposed DFRA methodology is compatible with recent trends in I/O forwarding adoption at large supercomputers, such as the Cray series.

For better I/O forwarding performance, Ohta et al. [50] present two optimization methods to reduce I/O bottlenecks: I/O pipelining and request scheduling. Vishwanath et al. [63] boost I/O forwarding through a work-queue model to schedule I/O and asynchronous data staging. PLFS adds an interposition software layer that transparently partitions files to improve N-1 performance [20]. DFRA is orthogonal to these optimizations and focuses on application-aware forwarding allocation and performance isolation.

Resource-aware scheduling This work echoes efforts in resource-aware scheduling, such as approaches improving utilization of datacenters/cloud resources, including CPU, cache, memory, and storage [22, 34, 58]. Our focus, however, is on HPC systems. To this end, AID [44] identifies applications' I/O patterns and reschedules heavy I/O applications to avoid congestion. CALCioM [28] coordinates applications' I/O activities dynamically via inter-application communication. Gainaru et al. propose a global scheduler [31], which based on system condition and applications' historical behavior prioritizes I/O requests across applications to reduce I/O interference. The libPIO [65] library monitors resource usage at the I/O routers and based on the loads allocates OSTs to specific I/O clients.

Regarding application-level I/O aware scheduling, ASCAR [40] is a storage traffic management framework that improves bandwidth utilization by I/O pattern classification. Lofstead et al. [46] propose an adaptive approach that groups processes and directs their output to particular storage targets, with inter-group coordination. IOrchestrator [71] builds a monitoring program to retrieve spatial locality information and schedules future I/O requests.

Our proposed scheme takes a different path that does not require any application or I/O middleware modification. It observes application and system I/O performance, and based on both real-time monitoring results and past monitoring history, automatically adjusts its default allocation to grant more or dedicated forwarding resources.

I/O interference analysis On detecting and mitigating interference, Yildiz et al. [70] examine sources of I/O interference in HPC storage systems and identify the bad flow control across the I/O path as a main cause. CALCiom [28] and Gainaaru's study [31] show that concurrent file system accesses lead to I/O bursts, and propose scheduling strategy enhancements. On relieving burst buffer congestion, Kougkas et al. [39] leverage burst buffer coordination to stage application I/O. TRIO [66] orchestrates application's write requests in the burst buffer and Thapaliya et al. [60] manage interference in the shared burst buffer through I/O request scheduling. The ADIOS I/O middleware manages interference by dynamically shifting workload from heavily used OSTs to those less loaded [42]. Qian et al. [52] present a token bucket filter in Lustre to guarantee QoS under interference.

This work is complementary to the above studies and uses interference analysis as a tool, achieving performance isolation using *interference avoidance*.

8 Conclusion

In this work, we explore adaptive storage resource provisioning for the widely used I/O forwarding architecture. Our experience of deploying it on the No.3 supercomputer and evaluating with ultra-scale applications finds dynamic, per-application forwarding resource allocation highly profitable. Judiciously applied to a minor fraction of jobs expected to be sensitive to forwarding node mapping, our remapping scheme both generates significant I/O performance improvement and mitigates inter-application I/O interference. We also report multiple prior findings by other researchers as confirmed or contradicted by our experiments. Finally, though this study has focused on the allocation of forwarding nodes, the same approach can apply to other resource types, such as burst buffer capacity/bandwidth allocation.

Acknowledgement

We thank Prof. Zheng Weimin for his valuable guidance and advice. We appreciate the thorough and constructive comments/suggestions from all reviewers. We thank our shepherd, Rob Johnson, for his guidance during the revision process. We would like to thank the National Supercomputing Center in Wuxi for great support to this work, as well as the Sunway TaihuLight users for providing test applications. This work is partially supported by the National Key R&D Program of China (Grant No. 2017YFA0604500 and 2016YFA0602100), and National Natural Science Foundation of China (Grant No. 61722208, 41776010, and U1806205).

References

[1] A description of the advanced research WRF version 3. http://www2.mmm.ucar.edu/wrf/users/.

[2] Cori supercomputer. http://www.nersc.gov/users/computational-systems/cori/.

[3] Cray burst buffer in Cori. http://www.nersc.gov/users/computational-systems/cori/burst-buffer/burst-buffer/.

[4] Cray data virtualization service (DVS). https://pubs.cray.com/content/S-0005/CLE%206.0.UP05/xctm-series-dvs-administration-guide/introduction-to-dvs.

[5] K supercomputer. http://www.aics.riken.jp/en/.

[6] Lightweight file systems. https://software.sandia.gov/trac/lwfs.

[7] MPI-IO test. http://freshmeat.sourceforge.net/projects/mpiiotest.

[8] Oakforest-PACS supercomputer. http://jcahpc.jp/eng/ofp_intro.html.

[9] Piz Daint supercomputer. https://www.cscs.ch/computers/dismissed/piz-daint-piz-dora/.

[10] Sequoia supercomputer. https://computation.llnl.gov/computers/sequoia.

[11] Sunway TaihuLight supercomputer. https://www.top500.org/system/178764.

[12] The computational plant. http://www.sandia.gov/~rbbrigh/slides/conferences/salinas-cplant-lci02-slides.pdf.

[13] The IOFSL project. https://www.mcs.anl.gov/research/projects/iofsl/about/.

[14] The ZeptoOS project. http://www.mcs.anl.gov/research/projects/zeptoos/.

[15] Titan supercomputer. https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/.

[16] Top 500 list. https://www.top500.org/resources/top-systems/.

[17] Trinity supercomputer. http://www.lanl.gov/projects/trinity/.

[18] Adiga, N. R., Almasi, G., Almasi, G. S., Aridor, Y., Barik, R., Beece, D. K., Bellofatto, R., Bhanot, G., Bickford, R., Blumrich, M. A., et al. An overview of the Blue Gene/L supercomputer. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2002).

[19] Ali, N., Carns, P., Iskra, K., Kimpe, D., Lang, S., Latham, R., Ross, R., Ward, L., and Sadayappan, P. Scalable I/O forwarding framework for high-performance computing systems. In *IEEE International Conference on Cluster Computing (CLUSTER)* (2009).

[20] Bent, J., Gibson, G., Grider, G., McClelland, B., and et al. PLFS: A checkpoint file system for parallel applications. In *Proceedings of Supercomputing* (2009).

[21] Bin Yang, Xu Ji, X. M. T. Z. X. Z. X. W. N. E.-s. J. Z. W. L. W. X. End-to-end I/O Monitoring on a Leading Supercomputer. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2019).

[22] Bindschaedler, L., Malicevic, J., Schiper, N., Goel, A., and Zwaenepoel, W. Rock you like a hurricane: Taming skew in large scale analytics. In *European Conference on Computer Systems (EuroSys)* (2018).

[23] Bingwei Chen, Haohuan Fu, Y. W. C. H. W. Z. Y. L. W. W.-W. Z. L. G. W. Z. Z. Z. G. Y. X. C. Simulating the Wenchuan Earthquake with accurate surface topography on Sunway TaihuLight. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2018).

[24] Braam, P. J., and Zahir, R. Lustre: A scalable, high performance file system. *Cluster File Systems, Inc* (2002).

[25] Cui, Y., Olsen, K. B., Jordan, T. H., Lee, K., Zhou, J., Small, P., Roten, D., Ely, G., Panda, D. K., Chourasia, A., et al. Scalable earthquake simulation on petascale supercomputers. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2010).

[26] Cui, Y., Poyraz, E., Olsen, K. B., Zhou, J., Withers, K., Callaghan, S., Larkin, J., Guest, C., Choi, D., Chourasia, A., et al. Physics-based seismic hazard analysis on petascale heterogeneous supercomputers. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2013).

[27] Daley, C. S., Ghoshal, D., Lockwood, G. K., Dosanjh, S., Ramakrishnan, L., and Wright, N. J. Performance characterization of scientific workflows for the optimal use of burst buffers. *Future Generation Computer Systems* (2017).

[28] Dorier, M., Antoniu, G., Ross, R., Kimpe, D., and Ibrahim, S. CALCioM: Mitigating I/O interference in HPC systems through cross-application coordination. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2014).

[29] Fang, J., Fu, H., Zhao, W., Chen, B., Zheng, W., and Yang, G. swDNN: A library for accelerating deep learning applications on Sunway Taihulight. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2017).

[30] 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., and Yang, G. The Sunway TaihuLight supercomputer: System and applications. *Science CHINA Information Sciences* (2016).

[31] Gainaru, A., Aupy, G., Benoit, A., Cappello, F., Robert, Y., and Snir, M. Scheduling the I/O of HPC applications under congestion. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2015).

[32] Giltrap, D. L., Li, C., and Saggar, S. DNDC: A process-based model of greenhouse gas fluxes from agricultural soils. *Agriculture, Ecosystems & Environment* (2010).

[33] Gunawi, H. S., Suminto, R. O., Sears, R., Golliher, C., Sundararaman, S., Lin, X., Emami, T., Sheng, W., Bidokhti, N., McCaffrey, C., et al. Fail-slow at scale: Evidence of hardware performance faults in large production systems. In *16th USENIX Conference on File and Storage Technologies (FAST)* (2018).

[34] Helgi Sigurbjarnarson, Petur Orri Ragnarsson, Y. V. M. B. Harmonium: Elastic cloud storage via file motifs. In *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)* (2014).

[35] Heng Lin, Xiaowei Zhu, B. Y. X. T. W. X. W. C. L. Z.-T. H. X. M. X. L. W. Z., and Xu, J. ShenTu: Processing multi-trillion edge graphs on millions of cores in seconds. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2018).

[36] Iskra, K., Romein, J. W., Yoshii, K., and Beckman, P. ZOID: I/O-forwarding infrastructure for petascale architectures. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)* (2008).

[37] Kay, J., Deser, C., Phillips, A., Mai, A., Hannay, C., Strand, G., Arblaster, J., Bates, S., Danabasoglu, G., Edwards, J., et al. The Community Earth System Model (CESM) large ensemble project: A community resource for studying climate change in the presence of internal climate variability. *Bulletin of the American Meteorological Society* (2015).

[38] Kim, Y., Atchley, S., and Shipman, G. M. LADS: Optimizing data transfers using layout-aware data scheduling. In *13th USENIX Conference on File and Storage Technologies (FAST)* (2015).

[39] Kougkas, A., Dorier, M., Latham, R., Ross, R., and Sun, X. H. Leveraging burst buffer coordination to prevent I/O interference. In *IEEE International Conference on E-Science* (2017).

[40] Li, Y., Lu, X., Miller, E. L., and Long, D. D. E. ASCAR: Automating contention management for high-performance storage systems. In *IEEE International Conference on Massive Storage Systems and Technology (MSST)* (2015).

[41] Liu, N., Cope, J., Carns, P., Carothers, C., Ross, R., Grider, G., Crume, A., and Maltzahn, C. On the role of burst buffers in leadership-class storage systems. In *IEEE International Conference on Massive Storage Systems and Technology (MSST)* (2012).

[42] Liu, Q., Logan, J., Tian, Y., Abbasi, H., Podhorszki, N., Choi, J. Y., Klasky, S., Tchoua, R., Lofstead, J., Oldfield, R., et al. Hello ADIOS: The challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience* (2014).

[43] Liu, Y., Gunasekaran, R., Ma, X., and Vazhkudai, S. S. Automatic identification of application I/O signatures from noisy server-side traces. In *12th USENIX Conference on File and Storage Technologies (FAST)* (2014).

[44] Liu, Y., Gunasekaran, R., Ma, X., and Vazhkudai, S. S. Server-side log data analytics for I/O workload characterization and coordination on large shared storage systems. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2016).

[45] Lofstead, J., Jimenez, I., Maltzahn, C., Koziol, Q., Bent, J., and Barton, E. DAOS and friends: A proposal for an exascale storage system. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2016).

[46] Lofstead, J., Zheng, F., Liu, Q., Klasky, S., Oldfield, R., Kordenbrock, T., Schwan, K., and Wolf, M. Managing variability in the I/O performance of petascale storage systems. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2010).

[47] Luu, H., Winslett, M., Gropp, W., Ross, R., Carns, P., Harms, K., Prabhat, M., Byna, S., and Yao, Y. A multiplatform study of I/O behavior on petascale supercomputers. In *Proceedings of the 24th International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC)* (2015).

[48] Moreira, J., Brutman, M., Castano, J., and Engelsiepen, T. Designing a highly-scalable operating system: The Blue Gene/L story. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2006).

[49] Oberg, M., Tufo, H. M., and Woitaszek, M. Exploration of parallel storage architectures for a Blue Gene/L on the TeraGrid. In *9th LCI International Conference on High-Performance Clustered Computing* (2008).

[50] Ohta, K., Kimpe, D., Cope, J., Iskra, K., Ross, R., and Ishikawa, Y. Optimization techniques at the I/O forwarding layer. In *IEEE International Conference on Cluster Computing (CLUSTER)* (2010).

[51] Petersen, T. K., and Bent, J. Hybrid flash arrays for HPC storage systems: An alternative to burst buffers. In *IEEE High Performance Extreme Computing Conference (HPEC)* (2017).

[52] Qian, Y., Li, X., Ihara, S., Zeng, L., Kaiser, J., Süss, T., and Brinkmann, A. A configurable rule based classful token bucket filter network request scheduler for the lustre file system. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2017).

[53] Riesen, R., Brightwell, R., Hudson, T., Hudson, T., Maccabe, A. B., Widener, P. M., and Ferreira, K. Designing and implementing lightweight kernels for capability computing. *Concurrency & Computation Practice & Experience* (2009).

[54] Roten, D., Cui, Y., Olsen, K. B., Day, S. M., Withers, K., Savran, W. H., Wang, P., and Mu, D. High-frequency nonlinear earthquake simulations on petascale heterogeneous supercomputers. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2016).

[55] Sadaf Alam, Nicola Bianchi, N. C. M. C. M. G. S. G. M. K.-C. M. M. P. C. P. F. V. An operational perspective on a hybrid and heterogeneous Cray XC50 system. In *Proceedings of Cray User Group Conference (CUG)* (2017).

[56] Sakai, K., Sumimoto, S., and Kurokawa, M. High-Performance and Highly Reliable File System for the K computer. *Fujitsu Scientific & Technical Journal* (2012).

[57] Schmuck, F. B., and Haskin, R. L. GPFS: A shared-disk file system for large computing clusters. In *1st USENIX Conference on File and Storage Technologies (FAST)* (2002).

[58] Sigurbjarnarson, H., Ragnarsson, P. O., Yang, J., Vigfusson, Y., and Balakrishnan, M. Enabling space elasticity in storage systems. In *Proceedings of the 9th ACM International on Systems and Storage Conference (SYSTOR)* (2016).

[59] Smith, R., and Gent, P. Reference manual for the Parallel Ocean Program (POP), ocean component of the Community Climate System Model (CCSM2. 0 and 3.0). Tech. rep., Technical Report LA-UR-02-2484, Los Alamos National Laboratory, Los Alamos., (2002).

[60] Thapaliya, S., Bangalore, P., Lofstead, J., Mohror, K., and Moody, A. Managing I/O interference in a shared burst buffer system. In *International Conference on Parallel Processing (ICPP)* (2016).

[61] Tsujita, Y., Yoshizaki, T., Yamamoto, K., Sueyasu, F., Miyazaki, R., and Uno, A. Alleviating I/O interference through workload-aware striping and load-balancing on parallel file systems. In *International Supercomputing Conference (ISC)* (2017).

[62] Vijayakumar, K., Mueller, F., Ma, X., and Roth, P. C. Scalable I/O tracing and analysis. In *IEEE/ACM Petascale Data Storage Workshop (PDSW)* (2009).

[63] Vishwanath, V., Hereld, M., Iskra, K., Kimpe, D., Morozov, V., Papka, M. E., Ross, R., and Yoshii, K. Accelerating I/O forwarding in IBM Blue Gene/P systems. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2010).

[64] Wallace, D. Compute node Linux: Overview, progress to date, and roadmap. In *Proceedings of Cray User Group Conference (CUG)* (2007).

[65] Wang, F., Oral, S., Gupta, S., Tiwari, D., and Vazhkudai, S. Improving large-scale storage system performance via topology-aware and balanced data placement. *Oak Ridge National Laboratory, National Center for Computational Sciences, Tech. Rep* (2014).

[66] Wang, T., Oral, S., Pritchard, M., Wang, B., and Yu, W. TRIO: Burst buffer based I/O orchestration. In *IEEE International Conference on Cluster Computing (CLUSTER)* (2015).

[67] Wang, Y., Liu, J., Qin, H., Yu, Z., and Yao, Y. The accurate particle tracer code. *Computer Physics Communications* (2017).

[68] Xiaohui Duan, Ping Gao, T. Z. M. Z. W. L. W. Z. W. X.-H. F. L. G. D. C. X. M. G. Y. Redesigning LAMMPS for petascale and hundred-billion-atom simulation on Sunway TaihuLight. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2018).

[69] Xu, W., Lu, Y., Li, Q., Zhou, E., Song, Z., Dong, Y., Zhang, W., Wei, D., Zhang, X., Chen, H., et al. Hybrid hierarchy storage system in MilkyWay-2 supercomputer. *Frontiers of Computer Science* (2014).

[70] Yildiz, O., Dorier, M., Ibrahim, S., Ross, R., and Antoniu, G. On the root causes of cross-application I/O interference in HPC storage systems. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2016).

[71] Zhang, X., Davis, K., and Jiang, S. IOrchestrator: Improving the performance of multi-node I/O systems via inter-server coordination. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2010).