Los Alamos NATIONAL LABORATORY EST.1943

LA-UR-18-22116 (Accepted Manuscript)

Enforcing End-to-end I/O Policies for Scientific Workflows using Software-Defined Storage Resource Enclaves

Davis, Marion Kei
Karki, Suman
Nguyen, Bao
Zhang, Xuechen

Provided by the author(s) and the Los Alamos National Laboratory (2018-11-13).

To be published in: IEEE Transactions on Multi-Scale Computing Systems

DOI to publisher's version: 10.1109/TMSCS.2018.2879096

Permalink to record: http://permalink.lanl.gov/object/view?what=info:lanl-repo/lareport/LA-UR-18-22116

Disclaimer:
Approved for public release. Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

RESEARCH LIBRARY

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2879096, IEEE Transactions on Multi-Scale Computing Systems

Enforcing End-to-end I/O Policies for Scientific Workflows using Software-Defined Storage Resource Enclaves

Suman Karki, Bao Nguyen, Joshua Feener, Kei Davis, and Xuechen Zhang, *Member, IEEE*

Abstract—Data-intensive knowledge discovery requires scientific applications to run concurrently with analytics and visualization codes executing in situ for timely output inspection and knowledge extraction. Consequently, I/O pipelines of scientific workflows can be long and complex because they comprise many stages of analytics across different layers of the I/O stack of high-performance computing systems. Performance limitations at any I/O layer or stage can cause an I/O bottleneck resulting in greater than expected end-to-end I/O latency. In this paper, we present the design and implementation of a novel data management infrastructure called *Software-Defined Storage Resource Enclaves* (SIREN) at system level to enforce end-to-end policies that dictate an I/O pipeline's performance. SIREN provides an I/O performance interface for users to specify the desired storage resources in the context of in-situ analytics. If suboptimal performance of analytics is caused by an I/O bottleneck when data are transferred between simulations and analytics, schedulers in different layers of the I/O stack automatically provide the guaranteed lower bounds on I/O throughput. Our experimental results demonstrate that SIREN provides performance isolation among scientific workflows sharing multiple storage servers across two I/O layers (burst buffer and parallel file systems) while maintaining high system scalability and resource utilization.

Index Terms—QoS, Software-Defined Storage, Scientific Workflows

1 Introduction

The increasing trend towards data-intensive computing has elevated data analytics to first class entities in today's high-performance computing (HPC) data centers [1], [2], [3], [4]. Disk-based file systems are proving increasingly inadequate in meeting the demands of data analytics. As such, data management systems are moving from post-processing of massive data sets to new approaches in which more simulation outputs are processed by analytics operating *in situ*, i.e., concurrently with the simulations being run [5], [6], [7], [8], [9], and these new workflows tend to be more sensitive to vagaries in I/O performance.

While the technology exists to implement scientific workflows, there is a critical barrier to keeping performance-sensitive users from using the techniques, namely the lack of *operational support for the enforcement of end-to-end QoS policies*, which dictate the performance of data-intensive scientific workflows. As an example, the molecular dynamics simulation code LAMMPS workflow [10] uses atomic bond computation (Bonds) and atomic symmetry analysis (Csym) for material crack detection [11]. Bonds consumes data produced by LAMMPS, and its output is further analyzed by Csym. Typically end-users can specify computing policies (e.g., the number of cores and/or the number of compute nodes) for these applications using PBS [12] or Slurm [13] job scheduling software in HPC systems. However, no I/O policies (e.g., minimum required I/O bandwidth between LAMMPS and Csym) can be enforced together with the compute policies, leaving the end-to-end performance of the LAMMPS workflow highly unpredictable in shared storage systems that are simultaneously used by multiple jobs across the data center [14]. Although management of compute resources is well understood and supported using job schedulers, techniques for supporting the enforcement of end-to-end I/O policies are still in their infancy for HPC storage systems. It is very challenging for job schedulers of HPC systems to provide such services for several reasons.

First, an effective performance interface and underlying infrastructure are lacking for QoS specification for I/O—most production HPC clusters do not provide such a QoS interface for users, rather, their storage services are limited to sharing storage capacity (I/O throughput) with best-effort service quality. As such, the I/O performance of an application running in such an environment is highly unpredictable because of resource contention on various components of HPC systems such as network [15], memory [16], and disk [17], where I/O throughput has been observed to vary by 2.5 times between two identical runs [17]. For HPC systems supporting a QoS interface, it is in the form of a simple service-level agreement (SLA), usually specified as the lower bound on I/O throughput [18] or shares [19]. *No existing HPC system provides enforcement of both*. The problem is that the lower bound is only relevant when the sum of lower-bound throughput of jobs is close to the current capacity of storage systems. If a system has spare capacity after satisfying I/O lower bounds, administrators have no knobs to allocate it to workflows whose execution time is more sensitive to I/O performance. Nor can it be allocated to other pipelines (e.g., those in scientific workflows) that have correlated functionalities and may require a similar amount of storage resources in the same or adjacent time intervals.

Second, enforcement of end-to-end policies for I/O pipelines requires both coordination and differentiated treatment across

- *S. Karki, B. Nguyen, J. Feener, and X. Zhang are with the School of Engineering and Computer Science, Washington State University Vancouver, Vancouver, WA, 98685. E-mail: {suman.karki, bao.nguyen, joshua.feener, and xuechen.zhang}@wsu.edu.*
- *K. Davis is with the Los Alamos National Laboratory, Mail Stop B287, Los Alamos, NM 87545. E-mail: kei.davis@lanl.gov.*

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

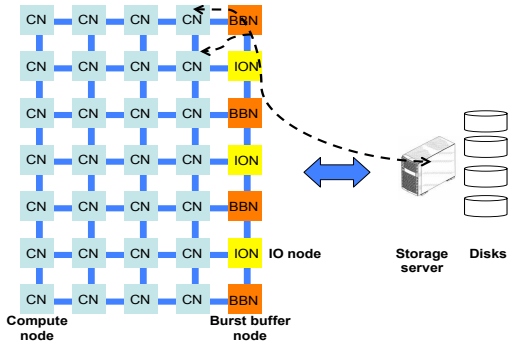

Fig. 1: Example I/O path of in-situ data analysis

multiple layers of I/O stacks. Figure 1 illustrates an I/O path for serving requests of data analytics on Cray machines at the National Energy Research Scientific Computing Center using a burst-buffer technique for in-situ execution [20]. On one hand, burst-buffer nodes, I/O nodes, and storage servers need coordinated operations to implement end-to-end policies. For example, to enforce the policy "minimum throughput of 200 MB/s between LAMMPS and Csym," all I/O layers along the I/O path need to be able to control the rate at which requests are processed. In addition, individual layers also need to recognize general high-level identifiers (e.g., "LAMMPS" and "Csym") to relate a request to the I/O pipeline and the policy that applies. On the other hand, because of differing semantics, individual layers need to treat requests differently. For example, I/O nodes handle requests in file semantics, while burst buffers might support other semantics (e.g., cache or memory semantics) for more efficient access to in-situ data.

Third, efficient implementation of QoS policies must account for the variable I/O demands of applications and the heterogeneity of storage devices without sacrificing system scalability and usability. Scientific applications are designed to aggregate I/O operations into dedicated I/O phases, such as loading data at the beginning of execution or writing periodic checkpoint data, so I/O requests can be very bursty. Background tasks [21] on both compute nodes and storage servers can also cause I/O throughput fluctuation. Without *dynamic reallocation of storage capacity* to applications, storage systems may be underutilized when an application that has a fixed reservation of storage resources is in a compute-intensive phase. Further, the impact of heterogeneity of storage devices is significant in the implementation of the policies. As an example, random access of disk is at least an order of magnitude slower than sequential access. Finally, HPC storage management systems need to provide an easy-to-use user interface that is compatible with existing software for CPU and memory management.

This paper proposes the concept of *software-defined storage resource enclaves* (SIREN) for resource management of HPC storage systems. SIREN provides a performance interface for users to specify I/O requirements such as *reservations* and *shares*. Reservations, expressed in absolute units, are lower bounds on I/O throughput requested by users. Shares specify relative allocation—the ratios in which different applications receive service—thus denoting I/O priority. SIREN allows users to set reservations and shares for individual applications or collectively for a group of related applications and analytics (e.g., the aforementioned LAMMPS, Bonds, and Csym). With SIREN administrators can partition storage resources hierarchically with the largest enclave spanning the entire storage system, and the smallest enclave serving a single analytic or simulation. Because multiple enclaves need to be managed in HPC systems, SIREN can allocate storage resources among enclaves with varied reservations and shares defined by users and serving workloads whose I/O demands dynamically change at runtime.

We have implemented a prototype of SIREN with a performance interface compatible with the semantics of PBS/TORQUE [22]. The performance interface is integrated with QoS-aware request schedulers on staging and I/O nodes of the OrangeFS parallel file system. Our evaluation with representative scientific workloads shows that SIREN is able to provide performance guarantees for single applications or a group of applications and dynamically allocate resources according to their I/O demands.

In summary, we make the following contributions.

- We propose a novel performance interface for I/O resource management in HPC systems. End users can use it to logically partition I/O resources among storage resource enclaves shared by a group of applications in a hierarchical manner. SIREN can guarantee resource reservations and shares for storage resource enclaves to enforce end-to-end I/O policies for performance-sensitive in-situ data analytics.
- A distributed scheduling algorithm is designed to effectively manage storage resource enclaves, which are organized as a tree. The algorithm takes end-users' QoS requirements, current I/O demands of applications, and enclave autonomy into account to provide performance guarantees, high resource usage, and software scalability.
- An enclave migration framework is designed to map enclaves to heterogeneous storage devices by considering how characteristics of devices and workloads affect system I/O efficiency. In this framework a machine-learning-based algorithm is used to automatically identify the opportunities for enclave migration. To reduce the migration overhead, SIREN supports dynamic mapping of logical file domains to physical locations.
- We have implemented SIREN at two key I/O layers—staging nodes and storage servers. Extensive evaluation of SIREN with real-world scientific workflows (e.g., GTS and LAMMPS) shows that it can effectively implement performance goals in terms of throughput reservations and shares with strong isolation among enclaves sharing burst buffers and parallel file systems.

The rest of the paper is organized as follows. Section 2 presents a motivating example for the needs of storage resource enclaves and discusses related work. Section 3 describes the design of resource partitioning algorithms and enclave mapping algorithms used in SIREN. Section 4 discusses SIREN's implementation in HPC systems and related issues. Section 5 describes and analyzes experimental results, and Section 6 concludes.

2 Motivation and Related Work

In this section we first present a motivating example to demonstrate the needs of storage resource enclaves, and then review the limitations of existing systems.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2879096, IEEE Transactions on Multi-Scale Computing Systems

Technique	*Distributed*	*Reservation*	*Share*	*Hierarchical*	*Multi-layers*
Centralized proportional sharing(SFQ)	No	No	Yes	No	No
PARDA [23]	Yes	No	Yes	No	No
Storage Resource Pools [21]	Yes	Yes	Yes	Yes	No
IOFlow [24]	Yes	No	Some	No	Yes
SIREN	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

TABLE 1: Comparison of SIREN with existing major resource allocation schemes.

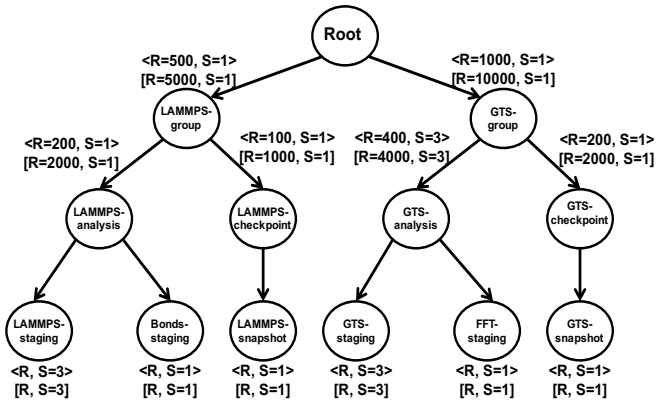

Fig. 2: An example of storage resource enclaves for in-situ data analysis sharing burst buffers and parallel file systems. R and S denote I/O reservations and shares respectively. $< R, S >$ and $[R, S]$ denote enclave QoS settings at the file system layer and at the burst buffer layer, respectively

2.1 The Need for Storage Resource Enclaves

A storage resource enclave is a set of storage resources dedicated to one application, or a group of applications together serving an I/O pipeline. Consider a consolidated HPC storage system as a resource enclave. Multiple workflows and background tasks are scheduled to run simultaneously on dedicated compute nodes sharing a pool of site-wide storage devices including I/O staging nodes and file system servers. In the following we use enclaves at the file system layer as an example. As depicted in Figure 2, the administrator sets up two first-level enclaves, *LAMMPS-group* and *GTS-group*, with settings reflecting the end-users' performance requirements for the workloads. There are four second-level enclaves, *LAMMPS-analysis*, *LAMMPS-checkpoint*, *GTS-analysis*, and *GTS-checkpoint*. The *LAMMPS-analysis* enclave manages resources for LAMMPS with Bonds being executed in situ for crack detection. The *GTS-analysis* enclave supports the GTS code with FFT being executed in situ for analysis of particles' spatial distribution while turbulence occurs. The other two enclaves, *LAMMPS-checkpoint* and *GTS-checkpoint*, manage I/O resources used by corresponding applications writing snapshots to parallel file systems.

The applications in the *GTS-group* enclave may need an overall reservation of 1000 MB/s. This total reservation is flexibly shared by its two sub-enclaves based on the peaks and troughs of demand during the execution of the GTS simulation. Within the same parent enclave, the ratio of shares of *GTS-analysis* and *GTS-checkpoint* enclaves is 3:1 reflecting a higher I/O priority for the applications running in the *GTS-analysis* enclave under contention. If an application (e.g., GTS) has low I/O demand, the storage resource should flow first to other applications in the *GTS-analysis* enclave before a different enclave, e.g., *LAMMPS-group*.

In summary, the semantics of storage resource enclaves enable rich control of end-to-end I/O performance of multi-stage scientific workflows. In addition, it distributes parents' reservations and shares to their children while ensuring that each child gets at least its own reservation and no more than its demand. Thus it reduces the burden on administrators and end-users to dynamically set QoS goals to achieve maximum resource utilization.

2.2 Previous Work

In-situ data analysis promises significant benefits, and a lack of end-to-end I/O performance guarantees is one of the most critical areas to be addressed and much research work is focusing on it. The work related to SIREN is discussed following.

Staging Resource Management in HPC Systems. Both resource provision and scheduling have profound effects on I/O performance of in-situ applications. Provisioning staging resources is usually done by considering both users' expected I/O performance and data access patterns of applications profiled offline [6]. Though it is orthogonal to scheduling, an effective provisioning approach provides elastic resources for adaptive scheduling, especially under I/O contention. Previous work provides administrators knobs for controlling the placement of staging data and the amount of allocated resources based on demands of applications [5], [25], [26]. Recently, I/O Container was designed to automatically alleviate the I/O bottleneck detected during data staging [27]. DataStager resolves contention between application-level communications and middleware-level communications [28]. DataSpaces supports data staging in deep memory hierarchies [29]. However, none of these provide end-to-end performance guarantees. Furthermore, most of these previous works concerned resource provision and scheduling for stand-alone jobs, which may comprise multiple in-situ analytics. In contrast, SIREN provides users an I/O performance guarantee for jobs running concurrently with all other jobs in a data center, where storage contention between jobs dominate I/O issues [17], [30], [31].

QoS-based I/O Resource Allocation. Much of the existing work on QoS-based I/O resource allocation supports proportional sharing, reservation, and hierarchical resource management. Table 1 provides a summary of existing approaches for QoS-based resource allocation and their comparison with SIREN. Proportional sharing has been well studied, and these designs are based on fair-queuing algorithms for throughput allocation among multiple I/O streams [32], [33], [34]. To minimize contention between the streams, approaches such as Argon [35] and YouChoose [36] explicitly set the quanta of disk service time for each stream, batch a number of requests from a stream, and serve them together to amortize the interference overhead. Stonehenge [32] and Facade [37] adjust the queue depth of disk schedulers (e.g., CSCAN) to effect a tradeoff between QoS and disk efficiency. PARDA provides a distributed proportional-share algorithm to allocate resource for VMs running on a cluster of ESX hosts [23]. The mclock algorithm supports I/O reservations

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

to guarantee the lower bound of I/O throughput for applications running on VMs [38]. Most recently, resource pools were proposed for hierarchical resource management [39], [21]. However, these works only help to enforce QoS policies at a particular layer of the I/O stack. IOFlow was designed to provide QoS control across multiple I/O layers for virtualized storage systems [24] but cannot be directly applied to HPC storage systems. We believe that SIREN is currently the only system that enforces end-to-end policies across I/O layers in consideration of the characteristics of tightly coupled in-situ analytics.

QoS-aware Management of HPC Storage Resources. QoS-aware resource allocation for scientific applications has been primarily investigated at the layer of parallel file systems. For example, vPFS supports QoS-aware storage resource management by implementing a variant of the DSFQ algorithm to reduce synchronization overhead [19]. U-Shape was proposed to provide storage QoS and performance isolation under I/O contention by using derived throughput bounds as the performance interface to maximize storage efficiency under changing data access patterns of applications [18]. Most recently, libsos [30] was designed to provide asynchronous I/O service for applications sharing object-based storage systems such as Ceph [40]. Here applications are required to make a resource reservation before being served and libsos is responsible for assigning disk service times to applications, thereby reducing I/O contention by avoiding concurrent access by multiple applications in a specific time interval. SIREN also provides basic storage QoS (e.g., proportional sharing and I/O reservation) in HPC systems. Most importantly, it guarantees end-to-end I/O performance across multiple I/O layers. Another unique feature of SIREN is its provision for performance isolation of a group of tightly coupled applications rather than a single application, thereby providing better resource allocation to in-situ analytics that have correlated functionalities and I/O behavior.

3 The Design of SIREN

SIREN is a scheduling framework to implement storage resource enclaves, in part by partitioning I/O resources. All enclaves created by SIREN are logically organized as a tree with the root node denoting either the whole of the resources in the system or a partition of the resources managed by SIREN. Resource allocation is done recursively at each level of the tree. The key challenges in the design of SIREN are (1) providing reservations while not suffering resource under-utilization, (2) maintaining system scalability, and (3) achieving high I/O efficiency in heterogeneous storage systems. To improve resource utilization SIREN dynamically distributes I/O resources according to reservations and shares of enclaves and the I/O demands of applications which are tracked online. For better scalability, SIREN provides autonomy to sub-enclaves and uses a publisher/subscriber communication model for tracking system statistics. To improve I/O efficiency, it migrates enclaves between heterogeneous storage devices according to the characteristics of workloads. In this section, we discuss the software architecture of SIREN and the distributed algorithms for its enclave management.

3.1 Overview of System Architecture

The SIREN scheduling framework is designed to manage storage resource enclaves. Figure 3 shows the overall system architecture. An HPC storage system contains multiple enclaves that are organized in a tree hierarchy. The enclave at the root manages

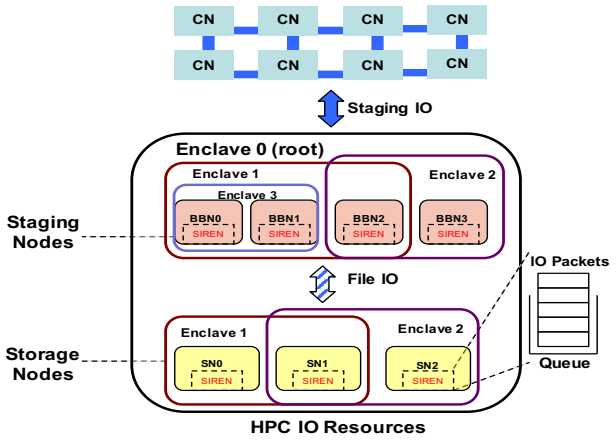

Fig. 3: SIREN controls resources allocation on staging nodes and I/O nodes using queues and QoS-aware I/O scheduling algorithms. *BBN*: staging nodes; *SN*: storage nodes.

all of the HPC I/O resources. The leaf enclaves directly issue requests to staging servers (e.g., burst-buffer nodes) or storage servers. Non-root enclaves are the basic scheduling instances of their parent enclaves. Thus an HPC storage system is recursively decomposed into multiple management domains. Resources may be shared by multiple enclaves (e.g., *BBN2* and *SN1*). Enclaves can have existence at multiple layers (e.g., *Enclave1* and *Enclave2*).

SIREN has two major components, the *enclave manager* and *request scheduler*, which work synergistically to achieve QoS goals in distributed storage clusters. The SIREN enclave manager is a user-level daemon that performs two major tasks: (1) aggregating I/O demand from its sub-enclaves at specified I/O layers, and (2) computing new values for the reservations and shares for use by its sub-enclaves considering the demands of all the layers. The SIREN request schedulers manage dispatching queues at various I/O layers to enforce the reservations and shares specified by the enclave manager. They periodically (1) receive the updated values of reservations and shares from leaf enclaves, (2) set the storage capacity after considering the data access patterns of workloads and divides it among the enclaves sharing the storage servers, and (3) schedule I/O packets in accordance with the allocations. The SIREN request scheduler is a variant of mclock [38]. Though it is similarly based on virtual time assignment for QoS scheduling, its virtual times are further adjusted based on I/O services received by other enclaves to achieve total-service proportional sharing in the whole storage system.

3.2 Hierarchical and Cross-Layer Enclave Management

Cross-layer demand aggregation. Enclave managers perform I/O demand aggregation bottom-up from the leaf to root enclaves. SIREN compares the reservation to demand in each enclave to ensure that the reserved I/O bandwidth is no more than the demand of applications. Algorithm 1 describes the algorithm for demand aggregation. The inputs to the demand aggregation module of SIREN include statistics, e.g., average I/O latency and throughput, which are collected during the previous monitoring interval in leaf enclaves and system parameters, e.g., the number of leaf enclaves and total I/O capacity (in IOPS) of a storage layer, and the capacity ratio between two layers. We use the approach of the Pesto system to dynamically determine the I/O capacity of a storage layer [41].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2879096, IEEE Transactions on Multi-Scale Computing Systems

Algorithm 1: Demand computation in enclave trees across I/O layers

Input: N_{leaf}: number of leaf enclaves;
$Capacity^i$: total capacity (IOPS) of layer i;
α^i: capacity ratio between layer i and $i+1$;
$latency^i_{leaf_j}$ and $IOPS^i_{leaf_j}$: average I/O latency (ms) and throughput (IOPS) of $enclave_j$ for $layer_i$;
τ_c: congestion threshold.
Output: $demand^i_{internal_k}$: demand of enclave k of layer i.

```
// Under-provisioned capacity for layer_i
1  capacity^i_under = 0
// Demand of enclave_j in terms of number of
   outstanding I/Os for layer_i
2  demandOIO^i_leaf_j = 0
// Update demands at each I/O layer
3  for layer i in 1, ..., n do
4      demandOIO^i_leaf_j = latency^i_leaf_j * IOPS^i_leaf_j
       // Update demands of leaf enclaves
5      demand^i_leaf_j = demandOIO^i_leaf_j / τ_c + capacity^(i-1)_under
6      demand^i_total = Σ_{j=0}^{m} demand^i_leaf_j
7      if demand^i_total > Capacity^i then
8          if i < n then
9              capacity^i_under =
               α^i * (demand^i_total − Capacity^i)/N_leaf
10         else
11             break;
       // Update demands of internal enclaves
12     for each internal enclave k in layer i do
           // n_c: the number of children of
              enclave k
13         demand^i_internal_k = Σ_{child=0}^{n_c} demand^i_child
```

For each enclave at a particular storage layer, SIREN computes I/O demand in terms of I/O throughput (in IOPS) using Little's law [42]. Specifically, it calculates the I/O demand $demandOIO^i_{leaf_j}$ in terms of outstanding I/O requests in dispatching queues using the average I/O latency and throughput measured in each enclave (line 4). Then $demandOIO^i_{leaf_j}$ is converted to I/O demand $demand^i_{leaf_j}$ quantified in IOPS for the convenience of calculation (line 5). Then we compute the total demand $demand^i_{total}$ by aggregating all the demands of leaf enclaves in layer i (line 6). If the total demand is larger than the capacity of storage layer i there is resource under-provisioning. SIREN addresses short-term resource under-provisioning by increasing the effective demands in layer $i+1$. In this scenario, SIREN quantitatively calculates how much more resource needs to be allocated to the next layer considering the capacity difference ratio α^i between the two layers and the amount of required resource in IOPS in layer i (lines 7-11). Finally, all the internal enclaves compute demands by aggregating demands of their child enclaves (lines 12-13). In the algorithm, τ_c is a congestion threshold determined based on our experimental results. For example, it can be set to 5 milliseconds and 10 microseconds for the SSD-based parallel file systems and burst buffers using DRAM, respectively.

Dynamic computation of reservations and shares. SIREN computes dynamic reservations and shares for enclaves at runtime based on the user-specified values of reservations and shares and the demand of enclaves from the root to leaf enclaves (top-down). The process can be seen in Algorithm 2. The inputs to the module include the user-specified shares, reservations, and demands of all the internal and leaf enclaves. While traversing all internal enclaves, SIREN computes the shares $share^{dynamic}_{ci}$ which are dynamically allocated to its child enclaves based on their respective weights computed using user-specified shares (lines 1-3). Then SIREN partitions an enclave's reservation among its children. If the aggregated reservation $reservation_{total}$ from the child enclaves is larger than its parent's reservation $reservation_{internal_j}$, the effective reservation $reservation^{dynamic}_{ci}$ allocated for each child is reduced according to its share weight (lines 6-7). Otherwise, $reservation^{dynamic}_{ci}$ is increased based on its weight (lines 8-9) since SIREN allocates the remaining reserved resource proportionally to its children according their shares. In the end, we need to compare $reservation^{dynamic}_{ci}$ to the demand of an enclave to determine if resource under-utilization may happen. If $reservation^{dynamic}_{ci}$ is higher than the demand, SIREN reduces it to the demand of the enclave. The complexity of the algorithm is $O(MN\log N)$, where M is the number of storage layers and N is the maximum number of enclaves in all layers. In our experiments M is equal to 2, consequently its complexity is $O(N\log N)$.

Algorithm 2: Recursively compute dynamic shares and reservations based on the enclave tree for each layer.

Input:
$share_{internal_j}$ and $reservation_{internal_j}$: user-specified share and reservation of internal enclave j;
C_i: Children of enclave j, $1 \le ci \le n$ with user-specified $reservation_{ci}$, $share_{ci}$ and its demand $demand_{ci}$.
Output:
$share^{dynamic}_{ci}$ and $reservation^{dynamic}_{ci}$: dynamic shares and reservation allocated to local schedulers.

```
// Compute total shares
1  share_total = Σ_{ci=1}^{n} share_ci
// Compute weight of children using
   user-specified shares
2  weight_i = share_ci / share_total
// Partition parent's shares among
   children
3  share^dynamic_ci = share_internal_j * weight_i
// Partition parent's reservations among
   children
4  reservation_total = Σ_{ci=1}^{n} reservation_ci
5  reservation_adjust = abs(reservation_total − reservation_internal_j)
6  if reservation_total ≥ reservation_internal_j then
7      reservation^dynamic_ci = reservation_ci -
       weight_i*reservation_adjust
8  else
9      reservation^dynamic_ci = reservation_ci +
       weight_i*reservation_adjust
// Update reservation according to demands
10 if reservation^dynamic_ci ≥ demand_ci then
11     reservation^dynamic_ci = demand_ci
```

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2879096, IEEE Transactions on Multi-Scale Computing Systems

Local request scheduling. The request scheduler has been implemented at two I/O layers to demonstrate the effectiveness of SIREN for enforcing shares and reservations across I/O layers. Specifically, it is integrated with ADIOS [43] I/O middleware to control the I/O throughput at burst buffers for in-situ execution, and in the request-scheduling module of the OrangeFS parallel file system [44].

3.3 Enclave Autonomy for Improving Scalability

SIREN must be a scalable resource management system. Centralized approaches (e.g., IOflow [24]) do not work in HPC systems having thousands of nodes. As such, SIREN provides enclaves a degree of autonomy to manage their own resources and adapt to changes in I/O demands of their sub-enclaves. Thus non-root enclaves can resolve local issues themselves without the need of high-overhead global coordination. In addition, it supports autonomy at multiple levels so that higher-level enclaves can make decisions less frequently and using less data.

More specifically, all non-root enclaves of SIREN update demand every four seconds. Demand variation greater than 10% of the total capacity managed by an enclave indicates a potential I/O issue in the system. The issue can be caused by dynamic data access patterns of applications or I/O contention among enclaves sharing the same physical servers. In both scenarios the enclave manager sends the current demand to its parent enclave manager. If the manager of the parent enclave X has enough capacity to satisfy the demand it will enable top-down computing of dynamic shares and reservations only for the subenclaves of X. If this is insufficient the demand will be successively forwarded to the parent enclave manager until the demand is satisfied or until reaching the root enclave. If the root enclave still cannot satisfy the demand SIREN reports a warning to system administrators indicating the need for external intervention.

3.4 Enclave Placement and Migration

HPC I/O systems usually consist of a collection of heterogeneous devices. For example, the staging nodes may contain solid-state disks (SSDs), while the storage nodes serve requests using disks or SSDs. We use a machine learning based approach to determine the initial placement of enclaves and their migration as the data access patterns of workloads change and I/O demand is updated.

3.4.1 Initial Placement of Enclaves

The goal of initial placement is to determine the *home* storage device based on conditions such as availability of storage space, device capacity, each enclave's required performance (e.g., throughput reservation). In addition, we need to consider its fitness for the storage device. For example, when an enclave has mostly sequential requests it could be served on both disk arrays or SSD arrays as long as the minimal throughput can be achieved. However, if an enclave has mostly random requests it is better served on SSD arrays or staging buffers.

We use a machine-learning-based approach to determine the fitness of an enclave with the respect to a specific storage device. Because the major factor that affects the fitness is data access pattern, SIREN extracts the pattern from sample requests issued from applications running in an enclave. Specifically, SIREN uses the following steps. (1) We train machine learning models to obtain the relationship between request service time and data access patterns of workloads (e.g., request size, type, number of processes, and sequentiality). The model is trained for each storage device using both a synthetic training data set and I/O traces profiled from the execution of representative applications. We use the CART model and Weka for the model training and testing [45], [36]. (2) We collect request samples of applications in the enclave from previous executions. If no samples are available, I/O traces generated by representative MPI/IO benchmarks are used in the calculation of the fitness of an enclave. The fitness is subjected to further adjustment at runtime as explained in Section 3.4.2. (3) We feed the sample requests of each enclave into the device's model to obtain the average service time of the sample requests for the enclave. The service time is then used by Algorithm 3 to determine the home device of the enclave considering other conditions, e.g., resource availability and throughput reservation.

Algorithm 3: Mapping enclaves to storage (*home*) devices based on the service time of their sample requests.

Input: ES: the set of all leaf enclaves;
$reservation_i$: throughput reservation of leaf enclave i;
$index_i$: home storage device specified by a user or by SIREN for the leaf enclave i;
$time_i^{ssd}$ and $time_i^{disk}$: service time of the sample requests of enclave i on SSDs and disks;
$Capacity^{ssd}$ and $Capacity^{disk}$: capacity of SSDs and disks.
Data: DS_{ssd}={}, and DS_{disk}={}: the sets of enclaves assigned to SSDs and disks.

```
// Sort enclaves in decreasing order of
   their throughput reservations
1  sort_dec(ES);
2  for i ∈ ES do
3      if index_i exists then
4          DS_{index_i} = DS_{index_i} ∪ i;
5      else
6          dev_index = find_min_index(time_i^ssd, time_i^disk);
7          total_capacity = Σ_{j∈DS_{dev_index}} reservation_j +
           reservation_i;
8          if total_capacity ≤ Capacity^{dev_index} then
9              index_i = dev_index;
10             DS_{dev_index} = DS_{dev_index} ∪ i;
11         else
12             index_i = disk;
13             DS_disk = DS_disk ∪ i;
14 if DS_ssd = ∅ or DS_disk = ∅ then
15     balance(DS_ssd, DS_disk);
16 return DS_ssd, and DS_disk;
```

For the initial mapping Algorithm 3 checks if the home device of an enclave has been specified by a user (lines 3-4). For example, if users specify that Enclave 3 (Figure 3) needs to run on SSDs, SIREN will not examine its fitness on other storage devices. If the home device is not specified, we deem the device that can provide the minimum service time for an enclave as its home device (lines 5-10). We use the CART model to calculate the service time of sample requests on each device. Finally, if the throughput reservation on the device is greater than its capacity, the enclave will be placed on the parallel file systems using disks (lines 11-13). In this scenario, its throughput reservation may not be enforced. If one of DS_{ssd} and DS_{disk} is empty and the other

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

is not, we randomly move one enclave from the non-empty set to the empty set to reduce the I/O interference on disks or SSDs and improve the load balance of the system (lines 14-15). In this paper we only consider the scenario of mapping enclaves to two devices: one SSD array and one disk array. The algorithm can be easily extended to support a storage system consisting of multiple heterogeneous storage arrays.

3.4.2 Enclave migration

As the data access patterns of workloads change and I/O demand is updated, the initial placement of enclaves as indicated by their home devices may become sub-optimal. SIREN needs to reevaluate the placement of enclaves periodically or when new jobs start executing on the cluster. It supports migrating enclaves between SSDs and disks to achieve high I/O resource utilization and efficiency. Specifically, it periodically runs Algorithm 3 to generate a new set of enclaves assigned to SSD arrays ds_{ssd} and disk arrays ds_{disk}. If $ds_{ssd} \neq DS_{ssd}$ or $ds_{disk} \neq DS_{disk}$, SIREN will set the migration flag. If the flag is set consistently, SIREN will check whether the identified new home device for an enclave has spare capacity to hold the enclave. If so, the enclave migration operations will be carried out. To reduce the migration overhead SIREN supports dynamic mapping of logical file domains to its physical locations. Therefore, it does not need to migrate data associated with applications to its destination storage device during migration.

Specifically, SIREN partitions a file into a series of logical file blocks of 4 MB. After migration, each logical block on the original home device is mapped to a physical data region on the new home device. Incoming write requests are directed to access the new home device directly, while read requests accessing the migrated file blocks are directed to access the latest version of the data according to the metadata recorded in the mapping table. A mapping table is implemented using the Redis key-value store [46] for request redirection. In this paper we assume the staging nodes are allocated only per users' requests. Therefore, enclave migration is only supported at the layer of parallel file systems. We leave enclave migration across multiple I/O layers as future work.

An example of enclave migration is shown in Figure 4. Assume the I/O throughput of 4KB sequential writes is 340 MB/s and 650 MB/s for the disk array on *SN0* and SSD array on *SN1*, respectively. Figure 4(a) depicts the initial mapping from the enclaves to heterogeneous storage devices. The enclaves of staging applications (e.g., *LAMMPS-staging*, *Bonds-staging*, *GTS-staging*, and *FFT-staging*) are served on staging nodes with NVRAM. The *LAMMPS-snapshot* and *GTS-snapshot* enclaves are served by *SN0* and *SN1* separately for minimal I/O interference on the devices. Because *LAMMPS-snapshot* has a lower throughput reservation than *GTS-snapshot*, it is placed on the storage nodes with hard disks and *GTS-snapshot* is placed on the nodes with SSDs. After the *Gerris-snapshot* enclave [47] is created for the simulation of complex interfaces between fluid phases, SIREN deems *SN1* to be the home device of *Gerris-snapshot* and *SN2* to be the home device of *GTS-snapshot* and *LAMMMPS-snapshot*. Therefore, *GTS-snapshot* is migrated from *SN1* to *SN0*. The new mapping is shown in Figure 4(b). If *GTS-snapshot* is not migrated SIREN cannot enforce the throughput reservation on *SN1* for enclaves *GTS-snapshot* and *Gerris-snapshot* because of resource under-provisioning.

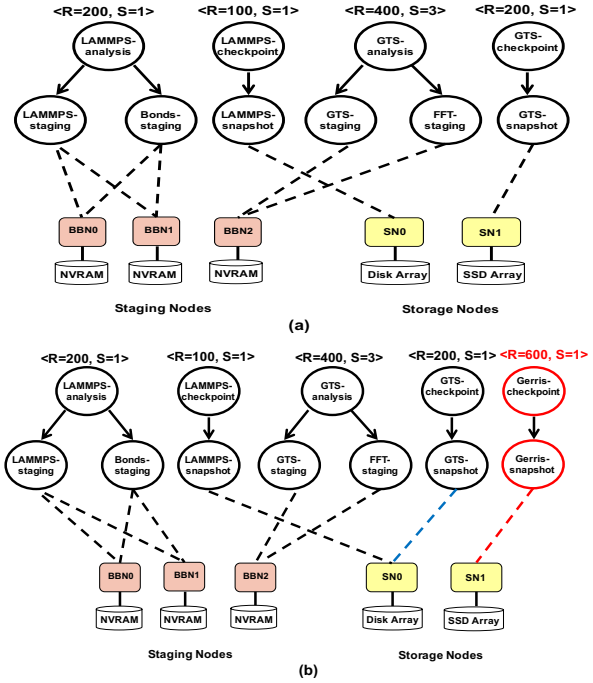

Fig. 4: Mapping of enclaves to staging nodes and storage nodes. (a) The initial mapping; (b) The mapping after a new enclave *Gerris-checkpoint* is created for the snapshot of Gerris. Because the new enclave has a higher reservation, the *GTS-snapshot* enclave is migrated from *SN1* to *SN0*.

```
#!/bin/bash -l
#PBS –l walltime=2:00:00, nodes=10:ppn=8
#PBS –l penclave=root:enclave=LAMMPS-group:
        reservation=500:share=1:file_list=atom,snapshot
#PBS –l penclave=LAMMPS-group:enclave=LAMMPS-analysis:
        reservation=200:share=2:file_list=atom
#PBS –l penclave=LAMMPS-group:enclave=Lammps-checkpoint:
        reservation=100:share=1:file_list=snapshot

mpirun –np 24 lammps < inputfile &
mpirun –np 2 bonds &
mpirun –np 2 csym &
```

Fig. 5: A PBS job script using SIREN's user interface for the creation of enclaves. SIREN supports new PBS options, e.g., penclave and enclave (highlighted in red).

4 Implementation Considerations

Building a prototype HPC storage system with support for storage resource enclaves required the development described following.

User interface. Most HPC users manage jobs using PBS for CPU and memory management [12]. Providing a PBS-compatible interface could significantly accelerate the adoption of SIREN. To that end we enhanced the PBS interface with new functions and semantics for storage resource management. Figure 5 shows an example of a PBS job script using the proposed enclave semantics for the creation of storage resource enclaves for LAMMPS and its related data analytics Bonds and Csym. Reservations set by users are given in the unit of MB/s. Its value is converted to IOPS based on average request size collected by SIREN at runtime.

Management plane. We use the BEACON library to im-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2879096, IEEE Transactions on Multi-Scale Computing Systems

plement the management plane of SIREN [48]. BEACON is a distributed, shared, out-of-band messaging bus used to transfer performance data (e.g., average I/O latency and throughput of leave enclaves) and control commands (e.g., dynamic reservations and shares) between enclave managers. It supports a publisher/subscriber interface and asynchronous communication resulting in minimal interference with applications at large scale. Section 5.8 studies the performance of the management plane.

Naming and metadata management. Enclaves are named using high-level identifiers. At every I/O layer enclave managers need to recognize enclave names and correctly correlate the names with identifiers in I/O requests (e.g., file handles for parallel file systems). In addition, the tree topology needs to be tracked from job creation to completion. For management of such metadata we use the Redis key-value store which provides scalable and low-latency access [46].

Response to Failures. Enclave managers may fail because of hardware and software failures, leading to temporarily degraded performance. To handle failures we build the following recovery mechanism. (1) Each enclave manager periodically sends a heartbeat message to its parent. (2) When an enclave failure is detected, the manager of the parent enclave creates a new enclave whose initial state is determined by the last recorded state of the failed enclave and current status of the storage system. (3) During recovery, a conservative default policy is used for resource allocation and I/O scheduling.

5 Evaluation and Analysis

We conducted an extensive performance study of SIREN to experimentally answer the following questions.

- How well does SIREN enforce QoS policies across workflow stages and storage layers?
- Is SIREN scalable for real scientific workflows with diverse data access patterns?
- What is the accuracy of the CART machine-learning model and can enclave migration be efficiently implemented?
- Is the management plane scalable to supercomputers?

5.1 System Setup

SIREN was prototyped and evaluated on the Kamiak cluster at Washington State University [49]. The cluster consists of 108 compute nodes, each of which is configured with a 28-core 2.0 GHz Intel Xeon E5-2660 processor, 128 GB DRAM, a heterogeneous storage system consisting of Intel Data Center S3520 480 GB SSDs and an enterprise-class disk array (10,000 RPM). Ten of the nodes were configured as data servers, one of which was also configured as the metadata server for the parallel file system. The SSDs on the data servers were used to set up a parallel file system using OrangeFS version 2.8.4 [44]. Files were striped over data servers with a 64 KB unit. We used the Flexpath transport [9] in the ADIOS [43] I/O middleware (version 1.12.0) to support in-situ data analysis. Ten compute nodes were set up as the staging nodes, which offer service similar to burst buffers. The compute processes are executed on dedicated compute nodes. All nodes share a 56 Gbs InfiniBand interconnect with 2:1 blocking for fast message passing in HPC applications. They run CentOS Linux release 7.3.1611 with kernel-3.10.0. We used MPICH2-3.2 with ROMIO to generate executables of MPI programs. To ensure that all data were accessed from the disk, the system buffer cache of each compute node and data server was flushed prior to each test run. We forced dirty pages to be written back every second on each data server.

In addition, we evaluated the scalability of SIREN on the Titan supercomputer at Oak Ridge National Laboratory [50]. Titan consists of 18,688 nodes, each of which is configured with a 16-core AMD Opteron 6274 CPU and 32GB memory. Each node runs Cray Linux Environment operating system. All nodes were interconnected with a Gemini network. Because installing the SIREN request scheduler on Titan may interrupt compute jobs, we focus on the scalability of its management plane built using BEACON in the evaluation.

5.2 Real-world Scientific Workflows

We used three scientific workflows in the experiments.

GTS workflow: The GTS petascale application is a Fortran particle-in-cell code designed for studying turbulent transport in magnetic fusion plasmas [51]. Multiple arrays are computed and written in its I/O phases for physics diagnostics, visualization, and/or checkpointing. In our experiments the GTS workflow consists of three stages: (1) *GTS-staging*: we rewrote multiple functions in the GTS code to use ADIOS for writing to burst buffers. For example, restart_write() is modified for staging the *zion* array, which contains the phase space coordinates of all the simulated *ion* particles. (2) *FFT-staging*: we instrumented a program called *FFT* analysis to read staging data from the burst buffer. It is one of the most heavily used analytics for GTS, calculating the particles' spatial distribution of elements in the *zion* arrays, and to execute Fourier transforms for physical diagnosis of the *phi* arrays. (3) *GTS-snapshot*: the staging data is written to the OrangeFS parallel file system for persistence. For the GTS workflow, we used 100, 10, and 100 processes for *GTS-staging*, *FFT-staging*, and *GTS-snapshot*, respectively.

LAMMPS workflow: LAMMPS is a popular Molecular Dynamics simulation code [10]. It is written with MPI and performs force and energy calculations on discrete atomic particles. In our experiments the LAMMPS workflow consists of three stages: (1) *LAMMPS-staging*: we instrumented the LAMMPS source code to use ADIOS for writing output to burst buffers. (2) *Bonds-staging*: the LAMMPS simulation is coupled with the *Bonds* [11] analytic for online data exploration. Bonds reads input from LAMMPS and performs all-nearest-neighbor calculations to determine which atoms are bonded together. It then further determines whether there are deformation zones in the material. If such zones are detected, it continues to calculate the conditions under which a crack occurred. (3) *LAMMPS-snapshot*: the staging data is written to the OrangeFS parallel file system for persistence. For the LAMMPS workflow, we used 100, 2, and 100 processes for *LAMMPS-staging*, *Bonds-staging*, and *LAMMPS-snapshot*, respectively.

Gerris workflow: Gerris was developed for simulations of surface-tension-dominant multiphase flows. It is one of the most widely used open-source Computational Fluid Dynamic applications. In our experiments, the workflow consists of *Gerris-checkpoint* for writing output to file systems and *Gerris-vis* for visualization of the simulation output. For the Gerris workflow, we used 10 and 1 processes for *Gerris-staging* and *Gerris-vis* respectively.

We did not change the concurrency of applications unless otherwise explained. The three workflows shared the SSD-based parallel file system and DRAM-based burst buffer. In the experiments, we set the capacity of the file system to 1.5 GB/s and

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2879096, IEEE Transactions on Multi-Scale Computing Systems

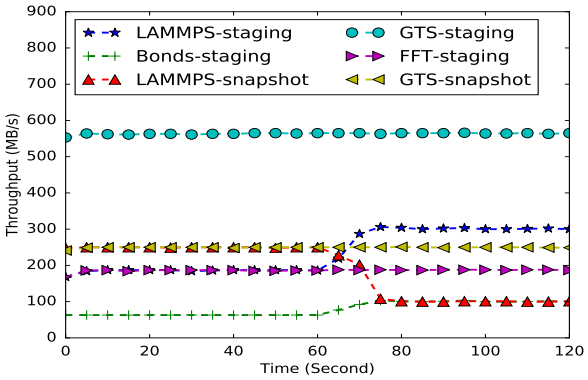

Fig. 6: Enforcement of QoS policies over 120 seconds of the simulations at the file system layer. The reservation for *LAMMPS-analysis* was increased from 200 MB/s to 400 MB/s after 60 seconds run time.

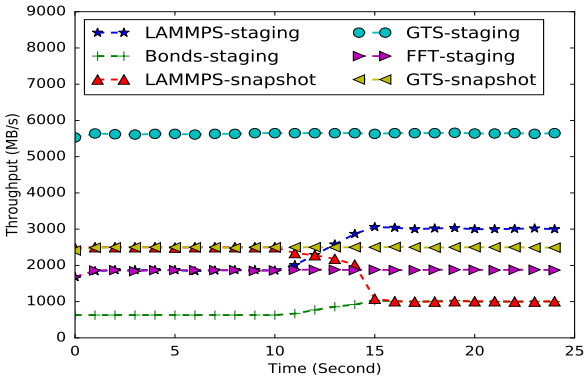

Fig. 7: Enforcement of QoS policies over 24 seconds of the simulations at the burst buffer layer. We increased the reservation of *LAMMPS-analysis* from 2000 MB/s to 4000 MB/s at time 10 seconds.

the burst buffer to 15 GB/s. Note that the capacity reserved for our experiments is less than the real system capacity and the remaining capacity is available to serve other HPC jobs.

5.3 Enforcement of QoS Policies across Stages

In this section we present three experiments that show the effectiveness of SIREN for enforcement of QoS policies across multiple stages of scientific workflows and demonstrate how SIREN provides performance isolation between enclaves. The initial resource control setting at each enclave is shown in Figure 2.

We measured the resource allocation at the layer of parallel file systems. Figure 6 shows the throughput allocated to each leaf enclave over 120 seconds of the simulations. With the initial setting, the average throughput allocated to the six leaf enclaves *LAMMPS-staging*, *Bonds-staging*, *LAMMPS-snapshot*, *GTS-staging*, *FFT-staging*, and *GTS-snapshot* are 187 MB/s, 63 MB/s, 250 MB/s, 563 MB/s, 187 MB/s and 250 MB/s respectively. The resource was allocated among *LAMMPS-group* and *GTS-group* according to their reservations because if the resource is allocated based on shares, *GTS-group* could get only 750 MB/s which is less than the minimum throughput 1000 MB/s as indicated by the reservation, resulting in an I/O bottleneck. After 60 seconds run time, we increased the reservation of *LAMMPS-analysis* from 200 MB/s to 400 MB/s. With the change, we can observe that (1) the throughput of the internal enclaves *LAMMPS-analysis* and *LAMMPS-checkpoint* changed from 250 MB/s to 400 MB/s, and 250 MB/s to 100 MB/s, respectively, and (2) the transition time is 16 seconds on average. The transition time depends on the number of outstanding I/O requests remaining in the dispatch queues. At time 76 seconds, in response to the change in the parent enclaves, the throughput of *LAMMPS-staging*, *Bonds-staging*, and *LAMMPS-snapshot* changed to 300 MB/s, 100 MB/s, and 100 MB/s, respectively.

In the second experiment we studied the enforcement of QoS policies at the burst buffer layer. Specifically, we implemented the QoS-aware burst buffer layer using the Flexpath transport in the ADIOS software. We ran the two workflows with the initial enclave QoS as shown in Figure 2. At time 10 seconds we increased the reservation of the enclave *LAMMPS-analysis* from 2000 MB/s to 4000 MB/s. We observe that (1) SIREN can enforce the QoS policy at the burst buffer layer, (2) the execution time of the two workflows are reduced by up to 80% because DRAM-based burst buffers provide higher I/O bandwidth, and (3) the transition time is 5 seconds which is much smaller than that at the file system layer because of the smaller queue length.

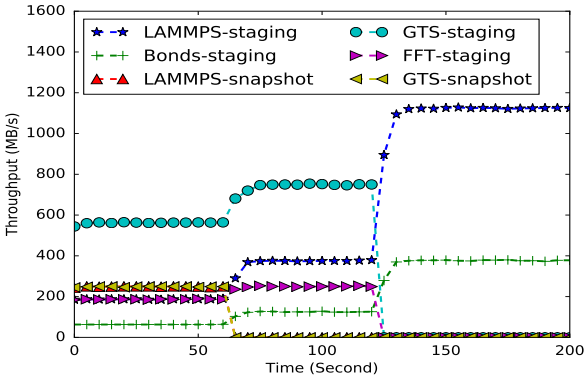

Fig. 8: Performance isolation. At time 60 seconds we stopped the execution of *LAMMPS-snapshot* and *GTS-snapshot*. Then at time 120 seconds, we stopped the execution of *Bonds-staging* and *FFT-staging*.

In the third experiment we studied whether SIREN can provide performance isolation among enclaves. The results are shown in Figure 8. At time 60 seconds we stopped the execution of *LAMMPS-snapshot* and *GTS-snapshot*. We observe that resources occupied by the enclaves are released and flow to its sibling enclaves. The throughput allocated to the two enclaves was increased from 187 MB/s to 375 MB/s, and 63 MB/s to 125 MB/s, respectively, according to the ratio of their shares. At time 120 seconds we stopped the execution of the applications in the enclave *GTS-group*. After 16 seconds, all the resources flowed to *LAMMPS-group*. Because only two applications *LAMMPS-staging* and *Bonds-staging* were running, their I/O throughput increased to 1125 MB/s and 375 MB/s, respectively.

5.4 Enforcement of QoS Policies across I/O Layers

SIREN enforces QoS policies across I/O layers to help achieve end-to-end performance of scientific workflows executed in situ. In addition, when I/O interference occurs abruptly at one I/O layer, it can immediately respond to these events by allocating

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2879096, IEEE Transactions on Multi-Scale Computing Systems

Enclave	Reservation (Burst buffer)	Reservation (File system)	Share (Burst buffer/ File system)
GTS-group	10000 MB/s	1000 MB/s	1 / 1
LAMMPS-group	5000 MB/s	500 MB/s	1 / 1
GTS-analysis	4000 MB/s	400 MB/s	3 / 3
GTS-checkpoint	2000 MB/s	200 MB/s	1 / 1
LAMMPS-analysis	2000 MB/s	200 MB/s	2 / 2
LAMMPS-checkpoint	1000 MB/s	100 MB/s	1 / 1

TABLE 2: QoS setting of internal enclaves for burst buffer and parallel file systems respectively. The setting of leaf enclaves is the same as that shown in Figure 2 at both layers.

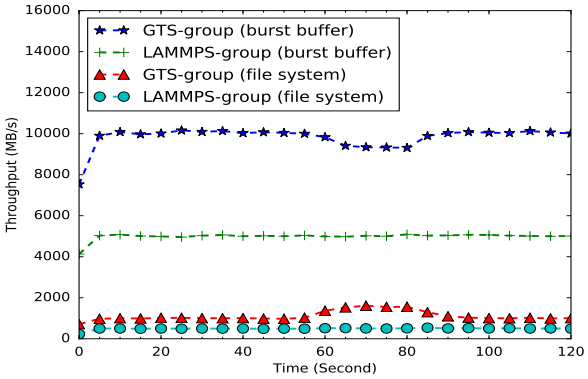

Fig. 9: Enforcement of QoS policies across the burst-buffer and parallel file system. The background job started at time 60 second causing I/O interference in the burst buffer.

more resource at other layers. In the experiments, we run the two workflows with the QoS setting as shown in Table 2 to control resource allocation at two I/O layers: a burst buffer with reserved capacity 15 GB/s and parallel file system with reserved capacity 1.5 GB/s. At time 60 seconds we run another staging application *VisIt* [52] to visualize the in-memory data produced by *FFT-staging* in the background for 20 seconds.

Figure 9 shows the aggregated throughput for the enclaves *GTS-group* and *LAMMPS-group* in the burst buffer and file system. We have two observations from the results. (1) When the I/O interference caused by *VisIt* occurred at time 60 seconds in the burst buffer, SIREN quickly responded by increasing the I/O demand of *GTS-group* at the file-system layer, resulting in more resource being allocated to serve the requests from the burst buffer for the GTS-group enclave. As a consequence, the execution time of the applications in the enclave did not change. (2) The performance of the LAMMPS workflow was not affected by the interference of the visualization application in the *GTS-group* enclave. This demonstrates that SIREN can provide performance isolation between enclaves even under I/O interference across multiple storage layers.

5.5 System Scalability

In this section we study the scalability of SIREN on Kamiak. We ran the two workflows and increased the number of processes of GTS and LAMMPS from 100 to 800. The QoS setting for the experiments is shown in Table 2. We measured the throughput of the enclaves *LAMMPS-group* and *GTS-group* and the aggregated throughput of the burst buffer and parallel file system with and without SIREN. The results are shown in Figure 10. When the

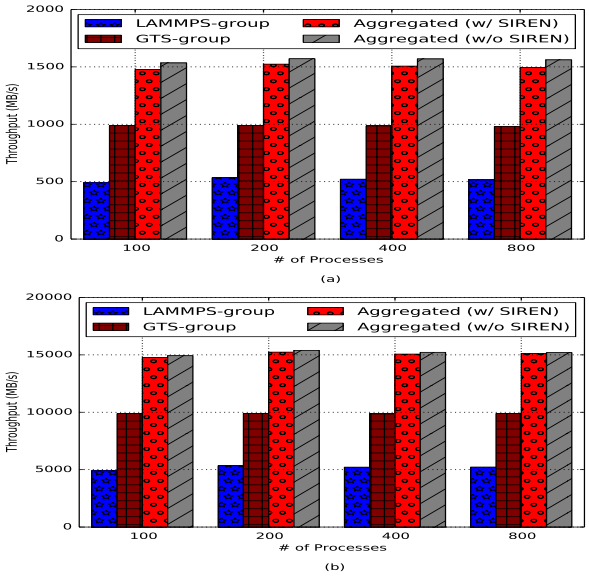

Fig. 10: Throughput with and without SIREN on Kamiak. (a) Throughput of the parallel file system; (b) Throughput of the burst buffer.

number of processes is increased from 100 to 800 we observe that SIREN can maintain the required throughput as indicated by reservations for both the burst buffer and parallel file system. For example, when the number of processes is 100, at the layer of parallel file system the throughput of the enclaves *LAMMPS-group* and *GTS-group* are 988 MB/s and 490 MB/s, respectively, which are very close to their reservations (1000 MB/s and 500 MB/s).

In addition, we measured the overhead of SIREN by comparing the system throughput without QoS control to control being enforced by SIREN. We found that the overhead is 0.9% on average at the burst buffer layer and 3.7% at the parallel file system layer. SIREN has a greater overhead in the parallel file system because the I/O latency of SSDs for random accesses is much greater than that of DRAM. At both of the layers, we also observed that there is no major fluctuation in the execution overhead as we increased the number of processes from 100 to 800, indicating that there is no scalability issue.

5.6 Accuracy of the ML Model

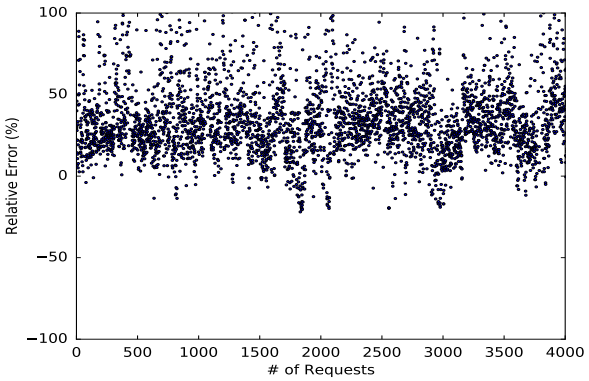

Fig. 11: Relative errors of I/O request service times of scientific applications.

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2879096, IEEE Transactions on Multi-Scale Computing Systems

ML Algorithm	*Random Forest*	*CART*	*M5P*
Relative Error	11.3%	23.7	32.2%

TABLE 3: The accuracy of the device models trained using different machine-learning algorithms.

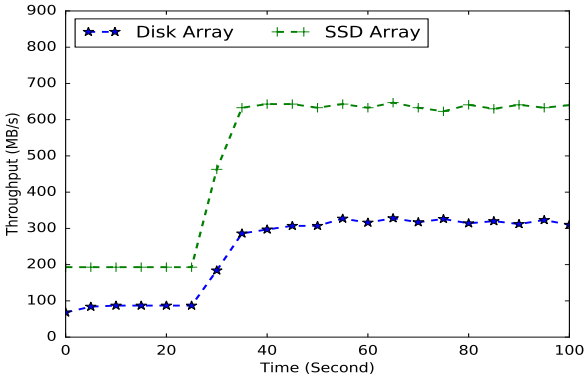

Fig. 12: Throughput of disk and SSD arrays before and after migration. The migration happens at time 25 seconds.

We evaluate the accuracy of the trained CART model of storage devices in this section. For training the CART model of the storage arrays and SSD arrays installed on Kamiak, we collect I/O traces when executing *GTS*, *LAMMPS*, and *Gerris* and trained each model using 200,000 I/O requests. In the fitness tests of the trained CART mode, we use 80% of the I/O traces as training data and the other 20% as a test data set to evaluate the prediction accuracy of the models. We use the relative error as the metric in the evaluation of the predication accuracy. Specifically, *relative percentage error* x is defined as $x = (T_m - T_p) * 100/T_m$, where T_m is the measured request latency (service time) and T_p is the predicted latency of the request by the CART model. The smaller the absolute relative error, the better the prediction. Figure 11 shows the relative errors for 4000 I/O requests from the three applications. From the results we observe that (1) the average relative error of all the requests is 23.7%, showing that the predicated latency is close to the measured latency, and (2) for a small number of requests the relative errors are greater than average.

We compared the accuracy of the storage-device models trained using three machine-learning algorithms, random forest [53], CART [54], and M5P [55] with results shown in Table 3. Of these, random forest achieves the highest prediction accuracy. The training time varies from 91 seconds with random forest to 120 seconds with CART. However, we found that the accuracy of the storage models is not sensitive to the prediction errors because we use the average latency of the sample requests in the determination of the mappings of enclaves to storage devices in Algorithm 3. The model training is executed offline.

5.7 Efficiency of Enclave Migration

A consolidated system may consist of heterogeneous storage devices. We evaluate the efficiency of enclave migration in a storage system consisting of one SSD array and one disk array installed on Kamiak. Enclaves were set up with the QoS requirements as specified in Figure 4. At time 25 seconds, the *Gerris-checkpoint* and *Gerris-snapshot* enclaves are created to serve snapshot I/O from the Gerris application. The initial mapping of enclaves and the mapping after migration are shown in Figures 4(a) and 4(b).

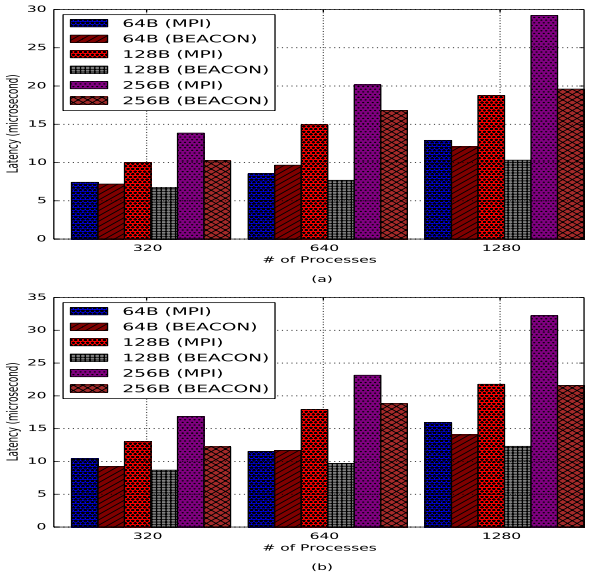

Fig. 13: Performance of SIREN's management plane on Titan with (a) 10 leaf enclaves and (b) 100 leaf enclaves.

In the experiment, we measure the I/O throughput of the SSD and disk array during the execution of the workflows. The results are shown in Figure 12. We have three observations. (1) Migration of enclave *GTS-snapshot* from *SN1* to *SN0* took 7 seconds. The execution of the proposed machine-learning module is 0.45 second with 5,000 I/O requests in the input. (2) After migration the heterogeneous storage system can serve three workflows compared to serving only two workflows without migration. (3) After migration the overhead of serving I/O requests is small because both of the devices can provide sustainable high throughput which is only 2.1% less than their capacity.

5.8 Performance of the Management Plane

We studied the performance of the management plane implemented using BEACON on the Titan supercomputer. In the experiments we used one root enclave and increased the number of leaf enclaves from 10 to 100. Every leaf enclave hosts one MPI program which employs one tenth of the N processes when the number of leaf enclaves is 10 and one hundredth of the N processes when the number of leaf enclaves is 100. Each process sends 1000 messages with performance data attached from the leaf enclaves to the root enclave. We implemented the communication plane using both BEACON and MPI and compared the message passing latency with different message sizes and numbers of processes in each leaf enclave. For the MPI implementation we use MPI asynchronous communication functions. For the BEACON implementation each process runs as a publisher and each enclave manager runs as a subscriber which asynchronously receives 1000 messages from publishers.

In the experiments we increased the total number of processes N from 320 to 1280 and message sizes from 64 to 256 bytes. The average latencies are shown in Figure 13. We observe that the latency with BEACON is 17.1% less than the latency with MPI because BEACON is a lightweight software system compared to the MPI library. For example, MPI needs to support various communication protocols, while BEACON currently uses only sockets. In addition, BEACON supports scoping to reduce communication overhead.

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2879096, IEEE Transactions on Multi-Scale Computing Systems

6 Conclusion

We have proposed a scheme, SIREN, to provide end-to-end I/O performance guarantees across multiple stages of a workflow using storage resource enclaves semantics. It hierarchically partitions I/O resources among enclaves by controlling the scheduling queue size using a distributed partitioning algorithm taking into account of both end-users' static settings (e.g., resource reservations and shares) and current I/O demands of applications. We integrated SIREN into two I/O layers: burst buffers and parallel file systems. Our evaluation with a diverse set of workloads shows that SIREN can faithfully and efficiently enforce end-to-end I/O policies specified by end-users, and provide strong performance isolation for in-situ analytics in different enclaves, with software overhead 1% and 3.5% in the burst buffer and parallel file system, respectively. SIREN supports enclave migration to achieve high I/O efficiency in heterogeneous storage systems.

In future work we will study the effect of QoS enforcement on the performance of PBS job schedulers and evaluate SIREN with other scientific workflows (e.g., S3D [56]) on Titan. To improve the accuracy of the storage models, we plan to use an incremental learning algorithm. We will further enhance SIREN to support enclave migration across multiple I/O layers.

Acknowledgments

We are grateful to the anonymous reviewers. This research was supported by US National Science Foundation under CRII ACI 1565338. This work was also funded in part by a WSU Vancouver Research Mini Grant and by the Exascale Computing Program of the US Department of Energy. Los Alamos National Laboratory is operated by Los Alamos National Security LLC for the US Department of Energy under contract DE-AC52-06NA25396.

References

[1] "Data-Intensive Computing Initiative," http://dicomputing.pnl.gov/.

[2] R. E. Bryant, "Data-intensive supercomputing: The case for DISC," in *Tech Report CMU-CS-07-128, Carnegie Mellon University School of Computer Science*, 2007.

[3] T. Hey and A. Trefethen, "The data deluge: an e-science perspective," *Grid Computing: Making the Global Infrastructure a Reality. Wiley*, 2003.

[4] A. S. Szalay, P. Z. Kunszt, A. Thakar, J. Gray, D. Slutz, and R. J. Brunner, "Designing and mining multi-terabyte astronomy archives: The sloan digital sky survey," in *SIGMOD International Conference on Management of Data (SIGMOD'00)*, 2000.

[5] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "Predata-preparatory data analytics on peta-scale machines," in *Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10)*, 2010.

[6] R. Prabhakar, S. S. Vazhkudai, Y. Kim, A. R. Butt, M. Li, and M. Kandemir, "Provisioning a multi-tiered data staging area for extreme-scale machines," in *Proceedings of the 2011 31st International Conference on Distributed Computing Systems (ICDCS'11)*, 2011.

[7] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*, 2012.

[8] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *In Proceedings of the 2012 IEEE Conference on Massive Data Storage (MSST'12)*, 2012.

[9] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, X. Zhang, H. Abbasi, S. Klasky, and N. Podhorszki, "Flexpath: Type-based publish/subscribe system for large-scale science analytics," in *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'14)*, 2014.

[10] S. Plimpton, R. Pollock, and M. Stevens, "Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations," in *Proc of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 2007.

[11] M. Wolf, Z. Cai, W. Huang, and K. Schwan, "Smartpointers: personalized scientific data portals in your hand," in *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'02)*, 2002.

[12] "Pbs batch script examples," https://www.olcf.ornl.gov/kb_articles/titan-batch-script-examples/.

[13] "The slurm job scheduler," http://www.arc.ox.ac.uk/content/slurm-job-scheduler.

[14] "The center-wide lustre file system," https://www.olcf.ornl.gov/kb_articles/spider-the-center-wide-lustre-file-system/.

[15] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: Performance degradation due to nearby jobs," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013.

[16] D. Skinner and W. Kramer, "Understanding the causes of performance variability in hpc workloads," in *Proceedings of the IEEE International Workload Characterization Symposium (IISWC'05)*, 2005.

[17] X. Zhang, K. Davis, and S. Jiang, "Iorchestrator: Improving the performance of multi-node i/o systems via inter-server coordination," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, 2010.

[18] X. Zhang, K. Davis, and S. Jiang, "Qos support for end users of i/o-intensive applications using shared storage systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, 2011.

[19] Y. Xu, D. Arteaga, M. Zhao, Y. Liu, R. Figueiredo, and S. Seelam, "vpfs: Bandwidth virtualization of parallel storage systems," ser. MSST '12, 2012.

[20] "Burst buffer at nersc," https://www.nersc.gov/users/computational-systems/cori/burst-buffer/.

[21] A. Gulati, G. Shanmuganathan, X. Zhang, and P. Varman, "Demand based hierarchical qos using storage resource pools," in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (ATC'12)*, 2012.

[22] "Torque resource manager," http://www.adaptivecomputing.com/products/open-source/torque/.

[23] A. Gulati, I. Ahmad, and C. A. Waldspurger, "Parda: Proportional allocation of resources for distributed storage access," in *Proccedings of the 7th Conference on File and Storage Technologies (FAST'09)*, 2009.

[24] E. Thereska, H. Ballani, G. O'Shea, T. Karagiannis, A. Rowstron, T. Talpey, R. Black, and T. Zhu, "Ioflow: A software-defined storage architecture," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013.

[25] F. Zheng, H. Zou, G. Eisenhauer, K. Schwan, M. Wolf, J. Dayal, T.-A. Nguyen, J. Cao, H. Abbasi, S. Klasky, N. Podhorszki, and H. Yu, "Flexio: I/o middleware for location-flexible scientific data analytics," in *27th IEEE International Parallel and Distributed Processing Symposium (IPDPS'13)*, 2013.

[26] T. Jin, F. Zhang, Q. Sun, H. Bui, M. Parashar, H. Yu, S. Klasky, N. Podhorszki, and H. Abbasi, "Using cross-layer adaptations for dynamic data management in large scale coupled scientific workflows," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13)*, 2013.

[27] J. Dayal, J. Cao, G. Eisenhauer, K. Schwan, M. Wolf, F. Zheng, H. Abbasi, S. Klasky, N. Podhorszki, and J. Lofstead, "I/o containers: Managing the data analytics and visualization pipelines of high end codes," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, 2013.

[28] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng, "Datastager: Scalable data staging services for petascale applications," in *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing (HPDC'09)*, 2009.

[29] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: An interaction and coordination framework for coupled simulation workflows," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010.

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2879096, IEEE Transactions on Multi-Scale Computing Systems

[30] M. J. Brim, D. A. Dillow, S. Oral, B. W. Settlemyer, and F. Wang, "Asynchronous object storage with qos for scientific and commercial big data," in *Proceedings of the 8th Parallel Data Storage Workshop (PDSW'13)*, 2013.

[31] Q. Liu, N. Podhorszki, J. Logan, and S. Klasky, "Runtime i/o re-routing + throttling on hpc storage," in *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'13)*, 2013.

[32] L. Huang, G. Peng, and T. Chiueh, "Multi-dimensional storage virtualization," in *the 2004 ACM SIGMETRICS Conference*. ACM, 2004.

[33] W. Jin, J. S. Chase, and J. Kaur, "Interposed proportional sharing for a storage service utility," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'04)*, 2004.

[34] Y. Wang and A. Merchant, "Proportional-share scheduling for distributed storage systems," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST'07)*, 2007.

[35] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger, "Argon: Performance insulation for shared storage servers," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST'07)*, 2007.

[36] X. Zhang, Y. Xu, and S. Jiang, "Youchoose: Choosing your storage device as a performance interface to consolidated i/o service," *Trans. Storage*, vol. 7, no. 3, Oct. 2011.

[37] C. R. Lumb, A. Merchant, and G. A. Alvarez, "Facade: Virtual storage devices with performance guarantees," in *Proceedings of the 2Nd USENIX Conference on File and Storage Technologies (FAST'03)*, 2003.

[38] A. Gulati, A. Merchant, and P. J. Varman, "mclock: Handling throughput variability for hypervisor io scheduling," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, 2010.

[39] V. Inc., "vsphere resource management guide:esx 4.1," in *ESXi 4.1, vCenter Server 4.1.*, 2010.

[40] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06)*, 2006.

[41] A. Gulati, G. Shanmuganathan, I. Ahmad, C. Waldspurger, and M. Uysal, "Pesto: Online storage performance management in virtualized datacenters," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing (SOCC'11)*, 2011.

[42] J. D. C. Little, "A proof for the queuing formula: L = λw," *Operations Research*, vol. 9, no. 3, 1961.

[43] "The adaptable io system (adios)," https://www.olcf.ornl.gov/center-projects/adios/.

[44] "Orangefs file system," https://www.orangefs.org.

[45] E. Frank, M. A. Hall, and I. H. Witten, "The weka workbench. online appendix for "data mining: Practical machine learning tools and techniques"," 2016.

[46] "Redis: An in-memory key-value store," https://redis.io/.

[47] "Gerris flow solver," http://gfs.sourceforge.net/wiki/index.php/Main_Page.

[48] "The backplane for event and control notification (beacon)," http://www.argo-osr.org/argo-software/.

[49] "Kamiak high performance computing cluster," https://hpc.wsu.edu/kamiak-hpc/what-is-kamiak/.

[50] "Titan supercomputer," https://www.olcf.ornl.gov/titan/.

[51] W. X. Wang, Z. Lin, W. M. Tang, W. W. Lee, S. Ethier, J. L. V. Lewandowski, G. Rewoldt, T. S. Hahm, and J. Manickam, "Gyro-Kinetic simulation of global turbulent transport properties in tokamak experiments," vol. 13, no. 9, 2006.

[52] "Visit: an open source, interactive, scalable, visualization, animation and analysis tool," https://wci.llnl.gov/simulation/computer-codes/visit/.

[53] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001.

[54] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger, "Storage device performance prediction with cart models," in *Proc. of the 12th MASCOTS Conference*, 2004.

[55] R. J. Quinlan, "Learning with continuous classes," in *5th Australian Joint Conference on Artificial Intelligence*. Singapore: World Scientific, 1992, pp. 343–348.

[56] J. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. Hawkes, S. Klasky, W. K Liao, k. l. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo, "Terascale direct numerical simulations of turbulent combustion using S3D," *Computational Science and Discovery*, vol. 2, p. 015001, 01 2009.

Suman Karki is a graduate student in the School of Engineering and Computer Science at Washington State University Vancouver. His research interests include file and storage systems and high-performance computing.

Bao Nguyen is a graduate student in the School of Engineering and Computer Science at Washington State University Vancouver. His research interests include file and storage systems and high-performance computing.

Joshua Feener is an undergraduate student in the School of Engineering and Computer Science at Washington State University Vancouver.

Kei Davis received his M.Sc. in Computation from the University of Oxford and his Ph.D in Computing Science from the University of Glasgow He is currently a research scientist at the Los Alamos National Laboratory, USA. His research interests include the areas of programming languages, operating systems, and parallel and high-performance computing.

Xuechen Zhang Xuechen Zhang received the M.S. and the Ph.D. in Computer Engineering from Wayne State University. He is currently an assistant professor in the School of Engineering and Computer Science at Washington State University Vancouver. His research interests include the areas of file and storage systems and high-performance computing. He is a member of the IEEE and ACM.

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.