2024 IEEE 31st International Conference on High Performance Computing, Data, and Analytics (HiPC) | 979-8-3315-0909-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/HIPC62374.2024.00031

Simulation of Large-Scale HPC Storage Systems: Challenges and Methodologies

Julien Monniot*, François Tessier*, Henri Casanova‡, Gabriel Antoniu*
*University of Rennes, Inria, CNRS, IRISA - Rennes, France
{julien.monniot, francois.tessier, gabriel.antoniu}@inria.fr
‡ University of Hawai'i - Honolulu, HI, USA
henric@hawaii.edu

***Abstract*—As the scale of production HPC platforms increases, so does the computing and I/O performance gap, exacerbating the storage bottleneck. High-performance storage systems have been developed to alleviate this bottleneck, but many questions remain concerning their architecture, implementation, and configuration. Answering these questions via experimental campaigns proves arduous. First, some answers are required before deploying the system. Second, once a system hits production the experimental scope is limited by the system's specific configuration and by constraints of production use. In this work we identify challenges posed by the design and validation of a storage simulator. We then propose solutions implemented in FIVES, a simulator of HPC workloads on platforms that comprise a parallel file system. We show how our simulator can be instantiated and calibrated for the accurate simulation of a production Lustre deployment.**

***Index Terms*—HPC, Storage, Modeling, Simulation**

I. Introduction

As increasingly powerful HPC systems are being deployed the gap between compute and I/O performance keeps widening. This is seen plainly by the evolution of the ratio between compute power and I/O bandwidth of the top three machines in Top500 [1], which has decreased by ∼10x over the last 13 years (see Figure 1). At the same time, the recent shift from compute-centric to data-centric applications and workflows has resulted in a so-called "data deluge", the effect of which has been observed in major supercomputing centers. For instance, at the National Energy Research Scientific Computing Center (NERSC) the volume of data stored by applications increased by ∼41x between 2010 and 2021, with an annual growth rate estimated at 30% [2].

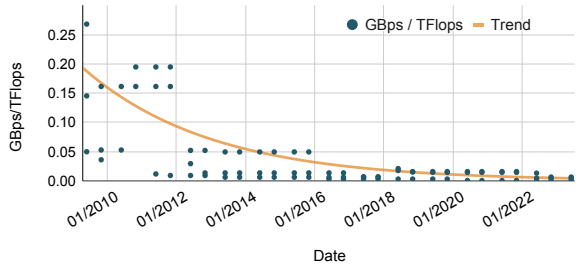


Fig. 1: Historical ratio of I/O bandwidth (GBps) to compute power (TFlops) of the top 3 systems of the Top500 list.

To alleviate the storage bottleneck, high-performance storage systems, vertically and/or horizontally scaled, have been installed alongside compute partitions. Vertical scaling consists in enhancing the storage infrastructure with additional fast storage layers based on flash memory or non-volatile memory [3]. This is done in emerging systems like DAOS [4] but also in mainstream parallel file systems like Lustre. Horizontal scaling consists in increasing the number of storage devices. For instance, this is the approach used by Orion [5], the 700PB parallel file system deployed on the Frontier supercomputer at Oak Ridge National Laboratory, which features 47,700 hard disks. The integration of I/O forwarding nodes into the interconnect also provides more gateways for I/O transit, reducing contention and increasing bandwidth. These developments have led to significant performance gains, at the cost of higher system complexity.

The architecture, implementation, configuration and efficient use of these high-performance storage systems open up many research questions. However, providing sound answers requires conducting comprehensive experiments, which often proves arduous. This is in part because some challenges, such as the sizing of the infrastructure, need answers upstream of the storage system design, before an actual system is available. One approach is to analyze job execution historical data, collected on other previously available systems. Unfortunately, this data is strongly tied to the particular features of those systems and of their application workloads. Even when a system is accessible for experimental purposes, the limitations imposed by its implementation and/or its production use reduces the scope of experiments that can be carried out to answer questions in areas such as storage-aware job scheduling, resource allocation, file system configuration or storage system's energy consumption. For instance, consider Lustre [6], the most widely deployed file system on supercomputers to date. The provisioning of a Lustre-based storage system, the choice of a data striping policy, or the development of strategies for mitigating contention are well-known questions that can only be answered via extensive experimental campaigns [7]. But running the necessary experiments at scale on a production system is a difficult proposition as these experiments would not only disrupt the system's production use, but could also lead to prohibitive resource and energy consumption.

Simulation is a promising approach to overcome the above

2640-0316/24/$31.00 ©2024 IEEE
DOI 10.1109/HiPC62374.2024.00031

experimental obstacles. It provides a way to 1) evaluate possible architecture designs and configurations before the system is deployed; 2) evaluate a wide range of I/O and storage resource management algorithms; and 3) perform post-mortem analysis of a decommissioned storage system and draw lessons from it. The simulation of parallel and distributed systems has long been an active field, enabling reproducible experiments for arbitrary scenarios in a way that is less time-, labor-, and resource-intensive than real-world experiments. However, there have been few attempts at simulating high-performance storage systems in a way that is *both accurate and fast*.

In this work, we first identify the main challenges for the design and validation of an accurate simulator of high-performance storage systems, ranging from the question of which data the simulator should take as input to the difficulty of achieving a desirable trade-off between simulation accuracy and simulation speed. We then propose solutions for these challenges, which we implement as part of FIVES, a "Simulator for Scheduling on Storage Systems at Scale" (five "S"). The goal of FIVES is to be sufficiently accurate for its output to inform file system configuration and design decisions. This goal is partly achieved via an automated simulation calibration method. Our contributions in this work are as follows:

- The identification of key challenges for the accurate and fast simulation of high-performance storage systems;
- A simulation abstraction for a distributed storage system;
- The design and implementation of the FIVES simulator;
- A method for automatically calibrating FIVES using Bayesian optimization;
- An experimental evaluation of FIVES for simulating a production Lustre deployment and workload.

II. SIMULATION CHALLENGES FOR HPC STORAGE

We have identified four key challenges for the accurate simulation of high-performance storage systems in HPC platforms. The first two are specific to this domain, while the last two are relevant to the simulation of parallel and distributed computing systems in general.

Challenge #1

Input data is hard to find, often incomplete or imprecise, and not usable out-of-the-box.

Assuming that an accurate high-performance storage system simulator is available, it must be provided input data that describes the workload to be simulated. Several "workload traces" have been collected on HPC platforms from various logs and produced by various monitoring tools (batch scheduler logs, I/O traces, parallel file system logs, ...). The goal is to combine these traces to construct a complete workload description that can be provided as input to the simulator. Unfortunately, not all necessary traces are always available for a single platform, and one often must work with incomplete data. Even if all the necessary data is available, it must be carefully curated because it contains artefacts due to incidents or misuses of the platforms, or to limitations or bugs of the monitoring tools [8].

Challenge #2

A simulator cannot be accurate for every I/O workload.

The heterogeneity of I/O workloads executed on real-world systems is high [9]. Furthermore, an inspection of actual application traces shows that many jobs are outliers, with behaviors (due to bugs, due to idiosyncratic implementations) that are hard to model and thus to simulate. This is because a program can implement I/O in many different ways, including ways that are vastly sub-optimal. In addition, even if we were to design a job model with enough flexibility to represent such a wide array of I/O behaviors, it would be very complex to benefit from it. Indeed, we don't have enough knowledge about the jobs to match each of them with a correct configuration of the model (see Challenge #1 above).

Challenge #3

The level of details of the simulation must be chosen to achieve a sensible accuracy/scalability trade-off.

A natural approach is to implement a simulator's simulation models with high levels of details in order to reproduce near-exact real-world behaviors. In many cases, such as the simulation of large HPC workloads and systems, doing so can lead to prohibitive time and space complexity due to large numbers of simulated discrete events. Furthermore, not all information is always available to implement highly detailed simulation models. For these reasons, one must instead resort to implementing less detailed, and thus likely less accurate, simulation models. The challenge is that for each component of the target system one must choose an appropriate level of detail at which to model this component, so that, overall, the simulator achieves sufficient levels of accuracy and of scalability.

Challenge #4

Calibrating a simulator with respect to ground-truth data[1] so as to maximize simulation accuracy is difficult.

All simulation models in a simulator come with configuration parameters, and appropriate values must be chosen for these parameters. Because simulation models are often not implemented at a high level of details, model parameter values cannot simply be picked based on the hardware specification of the system being simulated. For instance, say that a real-world network topology is abstracted away as a single macro-link. The latency and bandwidth values of this macro-link that maximize simulation accuracy are some complex functions of the latencies and bandwidths of the individual network links in the real-world network topology. In general, parameter values must be *calibrated* so that the simulator's accuracy is maximized with respect to available ground-truth data. This

[1]The reality of the modeled system behavior.

amounts to solving a multi-variate optimization problem in which the objective function is some simulation accuracy metric and the variables are the simulation model parameters. In the field of parallel and distributed computing, simulation calibration is often not performed and, when it is, it is mostly a labor-intensive and manual process [10].

To the best of our knowledge, there is no validated simulator of high-performance storage systems for which the above challenges have been addressed satisfactorily.

III. Related work

A. I/O and Storage Systems Analysis

The evaluation of high-performance storage systems has been actively pursued because, as applications' I/O requirements continue to grow, storage systems need to be studied and optimized throughout their lifetimes [2]. Multiple studies have focused on analyzing the behavior of specific storage system deployments over months of production usage [11], [12], [13], [14], [8], [15]. The goal of most of these works is to provide guidance to users and administrators of these specific deployments, without attempting to generalize to other hardware architectures or other low-level configurations of the system. They also require access to a significant volume of logs, often collected from multiple sources inside the studied systems (in [8], fives different logs were used). Other studies strive for more generalizable conclusions [16], [17], sometimes providing methods for tuning file system or application configuration parameters [7], [18]. These works, however, require extensive access to an already deployed file system, and are limited to configuration parameter tuning (no changes in the hardware infrastructure).

B. Simulation Approaches

Given the advantages of simulations over real-world experiments, it is not surprising that many parallel and distributed computing researchers have developed simulators, even though relatively few have focused on the simulation of distributed storage systems [19], [20], [21], [22].

Any simulator must implement simulation models that mimic the behavior of the components of actual systems. The most natural approach is to implement these models with a high level of details, in an attempt to reproduce "microscopic" real behaviors and achieve the highest possible accuracy (packet-level network simulation, cycle-accurate CPU simulation, block-level I/O device simulation). However, this is at the expense of scalability as the number of discrete events is typically proportional to the size of the workload being simulated. This scalability issue is problematic when wanting to simulate HPC workloads with large compute and data volumes. One solution is to use Parallel Discrete Event Simulation (PDES) [23] by which the simulation itself is a parallel application that executes on an HPC cluster [24], [25], [26]. This approach is used in [20], which, like this work, focuses on simulating a high-performance storage system. However, achieving high parallel efficiency with PDES is known to be challenging and, when simulating large systems and long-running workloads, the resource expenses for conducting extensive experimental campaigns can be prohibitive (in this work, for instance, we execute thousands of simulations).

The way to increase simulation scalability is to employ less detailed simulation models. These models aim to capture "macroscopic" behaviors of real-world systems, with time and space complexities orders of magnitude lower than those of the "microscopic" models discussed earlier. In the specific context of the simulation of high-performance storage systems, previous authors have developed simulators with such models [19], [21], [22]. These simulators are developed from scratch using a generic discrete-event simulation framework (namely SimPy [27]). However, considerable effort has been invested by the community in the development of discrete-event simulation frameworks for easing the development of simulators of parallel and distributed computing systems [28], [29], [30], [31], [32], [33], [34]. But these frameworks typically only provide simplistic simulation models of I/O resources and do not provide out-of-the-box solutions for the simulation of high-performance storage systems.

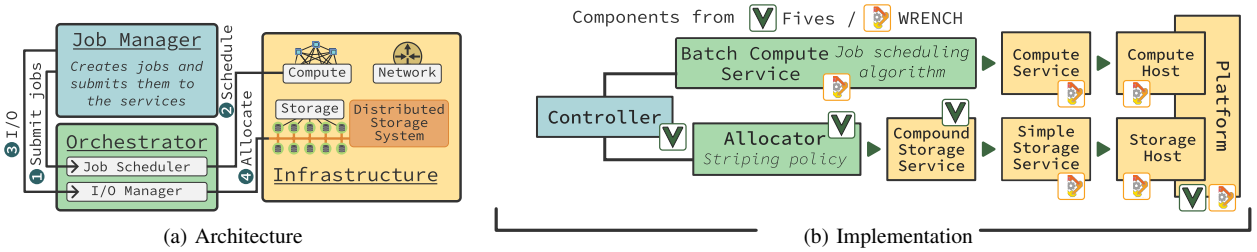
This work aims to address the challenges posed by the simulation of high-performance storage systems described in Section II. To do so we propose an architecture and an implementation of a storage system simulator called FIVES. In contrast to the aforementioned simulators, FIVES builds on (and contributes to) existing and well-established parallel and distributed computing simulation frameworks: WRENCH [33] and SimGrid [34]. We chose these frameworks because they are widely-used, provide the necessary simulation abstractions and models for this work, and have been the object of thorough validation studies that have demonstrated that they can achieve high accuracy [35], [36], [37], [38], [39], [40], [41], [33], [10].

IV. The FIVES Simulator

A. FIVES *conceptual architecture*

We consider that an HPC system comprises three main conceptual elements as depicted in Figure 2a: a *Job Manager*, an *Orchestrator* and an *Infrastructure*. The Job Manager receives user requests with compute resources and time demands, creates a job for each request, and submits jobs for execution to the Orchestrator. The Orchestrator implements scheduling policies by which jobs are assigned to particular hardware resources in the Infrastructure. We describe these conceptual elements in more details hereafter.

1) Infrastructure: The *Infrastructure* represents a simulated hardware platform, such as the one depicted in Figure 3. In this example the compute partition consists of homogeneous compute nodes interconnected via a Dragonfly network topology, and is defined by the number of compute nodes, their compute speed and number of cores, and the bandwidth and latency of the network links. Other network topologies are already available in SimGrid, and more can be manually implemented as needed. The storage partition comprises homogeneous storage nodes interconnected via a star topology, and is defined by the number of storage nodes and the bandwidth and latency



(a) Architecture (b) Implementation

Fig. 2: FIVES architecture and implementation.

of the network links. A disk is attached at each storage node, and is defined by a capacity, a read bandwidth, and a write bandwidth. The compute partition and the storage partition are connected to a common backbone interconnect via their network links. The backbone and links are defined by a bandwidth and a latency.

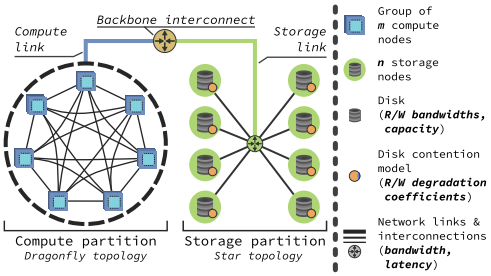


Fig. 3: FIVES's Infrastructure conceptual element.

2) Job Manager: The *Job Manager* is in charge of interpreting the resource demands in each user request, creating jobs, and sending these jobs to the Orchestrator. We define a *job* as a set of compute and I/O operations caused by the execution of one or more *applications*. These are to be executed on a set of resources on the *Infrastructure* (specified as a desired number of nodes and cores) that are requested for a given time period. A *reservation* corresponds to a job for which the requested resources have been allocated. During a *reservation* one or more *applications* may run, each with a known start and end timestamp within the bounds of the reservation.

3) Orchestrator: The *Orchestrator* is responsible for scheduling submitted jobs and their I/O operations. It thus must employ both a job scheduling algorithm and a striping policy for allocating storage resources and distributing data to disks. Although users can provide their own implementations of these algorithms, we have already implemented several algorithms in FIVES, described in the next section, which correspond to a large spectrum of relevant use cases (i.e., classical job scheduling algorithms, the striping policy of the Lustre file system).

B. FIVES Implementation

FIVES is implemented using the WRENCH [33] and SimGrid [34] state-of-the-art simulation frameworks. SimGrid provides foundational simulation abstractions for sequential concurrent processes that use compute, network, and storage hardware resources, using scalable and validated simulation models. WRENCH builds on top of SimGrid to implement high-level simulation abstractions of "Services" to which "Jobs" can be submitted and perform "Actions." Using these abstractions, simulators of complex workloads and systems can be implemented with minimal effort.

1) Main Simulation Components: Figure 2b depicts the simulation components used to implement the conceptual architecture described in Section IV-A, and shows the workflow of the simulation. The figure indicates which components are native to WRENCH and which are developed in FIVES.

The entry point component of FIVES is the *Controller*, which manages job executions throughout the simulation. It takes as input a dataset that specifies the jobs whose executions are to be simulated on the HPC platform, with an arrival date for each job. At the onset of the simulation, the *Controller* creates a set of jobs to simulate based on the job dataset. It submits each job for execution to the *Batch Compute Service* configured to use a particular job scheduling algorithm. WRENCH already comes with several such algorithms, and in this work we use its (default) conservative backfilling implementation [42]. The *Controller* acknowledges job completions until the execution of the entire workload has been simulated, and outputs timestamped job execution, I/O operation and I/O resource usage events. The *Controller* incurs no load during the simulation, but requires a few seconds before and after, while instantiating the jobs and computing the final metrics.

The jobs submitted by the *Controller* are executed on a simulated HPC system such as the one shown in Figure 3. SimGrid provides a powerful API for describing arbitrary hardware platforms to be simulated that includes compute, network, and storage resources. Using this API, the FIVES user can implement the *Infrastructure* conceptual element described in the previous section for any target hardware configuration. Building on top of SimGrid, WRENCH offers high-level services implemented in simulation that manages the simulated hardware resources: a *Compute Service* manages compute nodes (called *Compute Hosts*), while a *Simple Storage Service* manages storage nodes (called *Storage Hosts*). However, the *Simple Storage Service* implemented in WRENCH, which answers file read/write/delete/copy requests over the network, can only run on a single host and manage file systems on

disks attached to that host. To get around this limitation, which makes the framework unsuitable for modeling distributed storage systems, we introduce a new type of storage service that we have contributed to WRENCH: the Compound Storage Service (CSS).

2) Compound Storage Service: A CSS (see Figure 4) aggregates and provides a high-level interface to multiple Simple Storage Services on multiple hosts. Files stored on the CSS are transparently distributed and/or striped across any subset of the Simple Storage Services, and the CSS keeps track of the location of each file and file stripe. It intercepts requests for read, write, copy or delete operations in order to redirect them to the appropriate Simple Storage Services.

The CSS abstraction implements all core mechanisms for simulating a parallel file system. Custom policies, such as the striping policy, are provided through the *Allocator* component. This makes it possible to use the generic CSS abstraction and instantiate it to simulate a particular parallel file systems, as explained in Section V-B3.

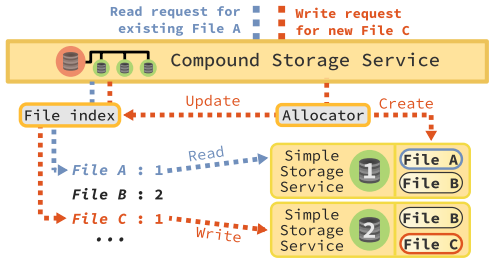


Fig. 4: CSS implementation in WRENCH

V. Use-case Driven Instantiation

Once a simulator has been implemented, it must be instantiated to be representative of a real-world system of interest. For this study, we selected Theta at the Argonne National Laboratory, a 11.7-PetaFLOPS Cray XC40 HPC system that ran for almost 6 years until the end of 2023, producing a significant number of major scientific results. Theta featured 4,392 compute nodes (Intel KNL) interconnected via an Aries network with a Dragonfly topology. The choice of Theta was motivated by its hosting of a 10 PB Lustre file system, the most widespread storage system on Top500 machines to date. In addition, unless explicitly specified at compile time, applications running on Theta were monitored with the Darshan [43] I/O monitoring tool, which has yielded a valuable dataset with years of I/O execution traces. An aggregated version of these traces is publicly available online [44].

A. Background

1) Darshan: Darshan [43] is a monitoring tool for recording the I/O performed by an application with low overhead. Information is collected at a high level of detail, down to the granularity of the data chunks written by a process. Darshan is deployed on several top-tier supercomputers, such as those at Argonne National Laboratory or at NERSC, where the majority of applications are monitored. However, for reasons specific to these institutions (confidentiality, data control), raw data is not publicly distributed. Only aggregated data is available, providing a global view of job I/Os but omitting information such as the number of files written or the number of processes that took part in I/O. Overall, for a given job, the information made available is job start and end times, volumes of data read and written, and time spent performing I/O (plus other information not relevant to this work).

2) The Lustre Parallel File System: Lustre is an open-source parallel file system (PFS) that has been maintained and evolved for over twenty years. As depicted in Figure 5, a Lustre file system consists of I/O servers called OSS (Object Storage Servers) and disks called OST (Object Storage Targets). The number of OSTs managed by an OSS is variable ranging from one on the Theta supercomputer at Argonne National Laboratory, to three on Frontier at Oak Ridge National Laboratory, or higher on other systems. Following the same model, metadata is managed by metadata servers (MDS) and targets (MDT). All storage resources are accessed via so-called LNET nodes, which correspond to the I/O forwarding nodes found in many HPC systems. These LNET nodes are generally part of the interconnection network and access the Lustre file system via a dedicated network interface.

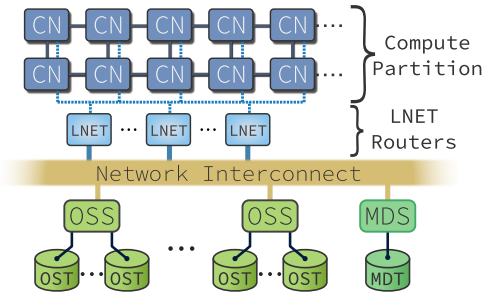


Fig. 5: Lustre architecture

Data written to a Lustre file system is striped across the OSTs, allowing performance gain by aggregating bandwidth from multiple OSTs and possibly OSSs. Striping a file is a two steps process. First, Lustre creates an ordered list of usable OSTs. Second, Lustre distributes file parts on a subset of this list according to a striping layout. Selecting and ordering which OST(s) will be used is achieved using one of two allocation strategies: round-robin or weighted. The round-robin strategy is used in most cases. The weighted strategy implements a bias towards selecting the least used OSTs first and is triggered on rare occasions, when free-space imbalance between OSTs is above some threshold.

Lustre's default striping layout takes two main parameters: the *stripe count* which defines the number of OSTs on which the stripes will be distributed and the *stripe size*, which defines the granularity at which the data is divided into chunks of identical size inside the stripes. The *stripe count* and the *stripe size* have default values, although they can be changed by

the user, on a per-file or per-directory basis. On Theta, the *stripe count* was set to 1 and the *stripe size* was set to 1MB. Section V-B3 describes our implementation of the Lustre's striping policy in the FIVES *Allocator* component.

B. Simulator Instantiation

Given the characteristics of the platform to be simulated and the information available regarding the jobs whose executions are to be reproduced on that platform, one must make various decisions regarding how the simulator should be instantiated. In what follows we discuss the three main decision axes when using FIVES to simulate the Theta platform and workload.

1) Job execution: When submitting a job to the *Batch Compute Service*, FIVES uses the real-world walltime and resource requirements of each *reservation* and respects the actual interval between subsequent job submissions, as available in the job trace. The start date of a job, however, is conditioned by the conservative backfilling job scheduling algorithm implemented in WRENCH. These start dates can thus be different from start dates that were driven by the specific batch scheduler configuration deployed on the real-world platform. Each job is defined by a runtime that we estimate based on the information present in the workload trace, and bound by the *reservation* walltime. Each *I/O phase* is subdivided in a variable number of individual read or write WRENCH actions, executed collectively from a subset of the job's compute nodes. Each action internally spawns multiple actual reads and writes to the storage system, depending on the job's I/O volume. The typical number of simulated reads or writes for a job can be approximated as $F \times H \times S_H$, where F ($1 \leq F \leq 10^2$) is the number of files, H ($1 \leq H \leq 10^1$) is the number of participating compute nodes and S_H ($1 \leq S_H \leq 10^2$) is the number of stripes accessed by each host, similar for all hosts ± 1. S_H is bounded by the number of OSTs in use and the number of file chunks allowed on each OST for simulation scalability reasons (see Section V-B3). This approximation is an underestimation of the number of I/O accesses that jobs performed on the real system. Despite this underestimation, our results demonstrate that our simulations are in line with real-world behaviors.

We assume that all reads and writes from a job are on the PFS. Some might be local since the Theta compute nodes feature node-local SSD, but the workload trace does not distinguish between local and non-local I/O. Furthermore, for a given job, the trace does not specify the number of files (F) and the number of compute nodes that are involved in I/O operations (H). We consider that F and H are parameters that must be calibrated based on observed real-world job executions. We assume that, for all jobs, the ratio of the total I/O volume to the total number of files, and the ratio of the number of compute nodes involved in I/O operations to the number of compute nodes allocated to the job, are based on fixed constants for reads and writes. In other words, we need to calibrate only 4 parameters (2 for reads, and 2 for writes), instead of $4n$ parameters where n is the number of jobs, which would render the calibration problem intractable due to high dimensionality. This choice leads to a smaller range of possible I/O behaviors among the simulated jobs, which at the moment we consider better than to allow more randomness, as our data doesn't currently provide us with details on the matter.

Challenge #1: Lesson learned

Making assumptions and abstracting away certain I/O behaviors is unavoidable because the information contained in I/O monitoring datasets is not comprehensive. Some of these assumptions, however, cause simulated workloads to be less heterogeneous than their real-world counterparts.

2) Disk contention model: SimGrid implements a default naive fair-share model for disk bandwidth, but users are advised to provide their own model for bandwidth degradation as a function of the number of concurrent I/O operations. This degradation depends on the mix of read and write operations and occurs both on HDDs and SSDs, with various behaviors that depend on many architecture- and workload-specific characteristics [37], [45]. Producing a general such model is outside the scope of this paper. Instead, we developed an empirical model derived from experiments we conducted on Seagate ST1000NX0443 SATA HDDs. The results (omitted due to space limitations) show that going from a single access to concurrent accesses causes an initial sharp decrease in bandwidth. But as concurrency increases, contention from each additional I/O operation has less impact and the bandwidth deterioration curve flattens out. It turns out that this behavior is modeled accurately (for our experimental results) using a model that includes a logarithmic component. Specifically, in FIVES, we model the instantaneous bandwidth (read or write) of a disk as:

$$bw = bw_{max} * \left(\frac{1}{C + \log n} \right) ,$$

where bw_{max} is the maximum achievable disk bandwidth, n is the number of ongoing concurrent I/O operations, and C is a constant. Difference choices for the C value makes the model more linear or more logarithmic. Values for bw_{max} and C must then be calibrated based on observed real-world job executions.

3) Striping Policy: The *Allocator* component provides the *Compound Storage Service* with a striping policy. Since Theta's storage system is a Lustre PFS, we have implemented in FIVES the internal striping policy based on Lustre's open source code. It handles both round-robin and weighted allocation strategies, with recommended default parameters for choosing between strategies, and accepts *stripe_size* and *stripe_count* parameters. It is important to note that the Darshan data source used in this study does not contain the *stripe_count* and *stripe_size* parameters used for each file. In our simulations, we resort to calibrating these values based on the following rules:

- *stripe_count*: All jobs with a mean I/O bandwidth (bw_{job}) below a calibrated threshold use the same static default value for all I/Os (set to 1 in this paper as it is the

default stripe count on Theta). Jobs above the threshold use a dynamic value computed as $alt\_stripe\_count \times bw_{job}/threshold$. The *alt_stripe_count* and the *threshold* values are different for reads and writes.

- *stripe_size*: This value is manually configured in the range [50, 100] MB. For simulation scalability reasons, this is higher than Lustre's default value of (1MB) to limit the number of simulation objects being created. For the same reason, FIVES can be configured so that the number of file parts on each OST is bounded by a user-specified values (F_{OST}). If using the *stripe_size* default value in conjunction with a large file size L ends up exceeding F_{OST}, the *stripe_size* is instead set to L / ($F_{OST} \times$ *stripe_count*) on a per-job basis. We pick F_{OST} so that one invocation of FIVES in our experimental evaluations takes at most 20min on a 2.5GHz core.

Challenge #3: Lesson learned

Each simulation model must abstract away some of the details of the real-world component it simulates for two reasons that can apply simultaneously: (i) because some of the details of the real-world system and/or workload are unknown and (ii) because of simulation scalability concerns.

VI. Simulator Calibration and Validation

Calibrating a simulator is essential for it to produce accurate results. The calibration process consists in determining simulation parameter values that make the simulated behavior as close as possible to the real-world behavior. We have identified several parameters to calibrate, as detailed in the next section. One way of determining values for these parameters is to vary each of them in successive simulation runs until a sufficient level of correlation between simulated and real performance is reached. However, this is a prohibitively long process due to the size of the parameter space and the non-zero simulation execution time. Instead, we use Bayesian Optimization (BO), a proven technique for parameter search that helps reduce the exploration space. In FIVES, we chose to work with the Ax framework [46], using 50-80 iterations.

A. Calibration parameters

FIVES is configured via dozens of parameters that pertain to the hardware platform, the jobs to be simulated, and the storage system. We have empirically determined which parameters play a significant role in the simulation output when simulating our production workload, leading us to identify 17 parameters that should be calibrated:

- Platform bandwidths (R/W on disks, network link between compute and storage partition) - 3 parameters
- Jobs file count (coefficient applied to job's mean bandwidth to determine read and write file counts for each I/O phase) - 2 parameters (R/W)
- Number of compute nodes participating in I/Os of each job - 2 parameters (R/W)
- Disk contention model coefficients - 2 parameters (R/W)
- Striping model parameters (stripe size and count values, adjusted per jobs and for R/W operations and metadata access overhead) - 7 parameters
- Maximum file parts count on OSTs - 1 parameter

Ranges of possible values for these parameters are loosely defined based on approximate knowledge of the platform, workload, and storage system. But for some parameters, as explained in Section V-B, the range is constrained for simulation scalability reasons.

Some of the parameters that are not calibrated are set to their known values (e.g., the total number of compute nodes, the number of disks per storage node). Others have been empirically determined to have little impact on the simulation output provided they are set to reasonable guesses. For instance, the network latency is set to the same value (24 μs) for all links, and only deviations from this value by orders of magnitude impact the simulation output.

1) Calibration loss function: The calibration process aims at minimizing a loss function that quantifies the simulator's accuracy. We define our loss function as the Mean Absolute Error of the percentage difference between the cumulative I/O time of each simulated job and its real-world counterpart. We use this percentage difference so that the impact of jobs with very long I/O on the loss function is not larger than that of jobs with shorter I/O. Formally, our loss function is defined as:

$$loss = \frac{1}{N} * \sum_{i=1}^{N} \frac{|R_i^{IO} - S_i^{IO}|}{R_i^{IO}} \ ,$$

where N is the number of jobs, R_i^{IO} (resp. S_i^{IO}) is the real (resp. simulated) I/O duration for job i.

B. Selecting calibration data

We find that our workload dataset includes either highly under-performing or over-performing jobs in terms of mean I/O bandwidth, e.g., ranging between 0.1MB/s and 3.5GB/s for reads. For our generic job model to be applicable, it is thus necessary to filter out outliers (mean read or write bandwidth $\geq$ 90$^{\text{th}}$ percentile or $\leq$ 10$^{\text{th}}$ percentile, I/O volume $>$ 10TB), which corresponds to $\approx 29\%$ of the jobs in the original dataset. The reason why we designed a generic job model, which cannot capture idiosyncratic behaviors of these outlier jobs, is that the workload dataset lacks the necessary information for attempting to model these behaviors. The remaining 82% of the jobs, however, can be simulated so that trends and correlations of simulated behaviors are in line with real-world behaviors. But still, we find that the calibration process is sensitive to the heterogeneity of the job dataset. Therefore, we sort jobs into bandwidth classes: *fast* jobs (mean bandwidth $\geq$ 75$^{\text{th}}$ percentile), *slow* jobs (mean bandwidth $\leq$ 25$^{\text{th}}$ percentile), and *regular* jobs in-between. Simulation parameters shared among all jobs (e.g., platform parameters) are calibrated using *regular* jobs only (and fixed for further calibrations). Job-specific parameters are calibrated independently for each class.

C. Validation Procedure

We use the obtained calibration to run simulations on every single month of the dataset, and evaluate simulation accuracy using Pearson correlations between simulated and real job I/O times. We are particularly interested in minimum values and variance in correlations for read and write times. The month that was used for obtaining the calibration serves as a control, the expectation being to get near perfect Pearson correlations between real and simulated I/O times. The other simulation runs let us observe whether the calibration generalizes, since the workload exhibit variations throughout the year.

Challenge #4: Lesson learned

Defining the set of calibration parameters is not straightforward as one must determine which parameters have a significant influence on the simulation output to reduce the dimensionality of the calibration problem. Choices must also be made regarding which subset of the ground-truth data should be used for computing the calibration, requiring some method to define and exclude outliers.

VII. Evaluation

In this section we quantify the simulation accuracy of the calibrated FIVES simulator. Direct comparison is not feasible between our results and that obtained with previously proposed simulators [19], [20], [21], [22], some of which have actually not been validated against real-world ground-truth data. For instance, neither [19] nor [22] offer a network and I/O bandwidth model which could be used as part of a feedback loop during the simulation, which make them fundamentally incompatible with FIVES. In [20], the authors present a simulator closer to FIVES, but which is able to model the execution of only a very limited number of applications at a time, and requires an unspecified, but consequent, cluster of nodes to execute. Due to these fundamental design and implementation differences (and others: supported input traces, simulated platform models, storage allocation algorithms, output metrics, ...), performing sound comparisons would require extensive modification of these previously proposed simulators to augment their existing capabilities.

A. Simulation calibration results

For calibration and validation of FIVES we consider one year of Darshan traces from the Theta system at Argonne. We choose the year 2022 because it is recent and representative of peak machine usage. Out of 18,086 individual jobs, ≈7,000 do not contain actual I/O activity, and we filtered out another ≈6,000 jobs based on the criteria defined in Section VI-B, leaving us with ≈5,000 suitable jobs. Computing the calibration using the full year as ground-truth data is computationally intensive, and our experiments show that it brings only small accuracy gains when compared to using a single month. In all results below, the calibration was computed using data from the month of November, considering only jobs in the *regular* class. We picked this month because it has a significant number of regular jobs (366 jobs) and is representative of the job heterogeneity seen in the full dataset. We use Pearson's correlation between simulated and real cumulative I/O times in order to report on the quality of the calibration. While this metric doesn't fully validate accuracy, it helps make sure that FIVES captures a satisfactory trend of the time spent in I/O, which is relevant for a simulation of large periods of time and thousands of jobs.

Figure 6 shows, for each job of the regular class from November 2022, the cumulative real I/O time (x-axis) extracted from Darshan traces and the FIVES-simulated cumulative I/O time (y-axis). This corresponds to the month that was used for calibrating the simulator, and we thus expect high accuracy. Most data points are close to the target diagonal, depicted as a red straight line, and the Pearson's correlation between simulated and real cumulative I/O times is 0.98.

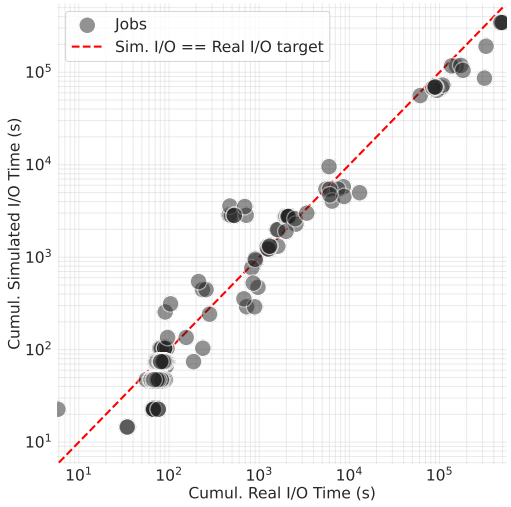


Fig. 6: Simulated vs. real cumulative I/O time for regular jobs of Nov. 2022 (366 jobs).

For the year of 2022, excluding the month of November, Figure 7 shows the I/O volumes (bytes) vs. the I/O time (seconds) for each real *regular* job (green) and its simulated counterpart (grey). We observe that simulated jobs almost all fall within the same job class as that of their real jobs counterparts (92% of jobs). Most of the simulated jobs falling outside of the class bounds form a horizontal pattern (dotted box in the figure). The same pattern, although contained within the class, is present in the real traces. Because these ≈ 200 jobs process almost exactly the same I/O volume and have mostly sequential job IDs, we conjecture that they come from the same application, executed multiple times under various platform conditions and/or with slightly different input parameters. However, the traces do not contain enough details to determine clear differences between these jobs, which is especially challenging for the calibration of FIVES (same, incomplete, input values for n jobs, but different expected outcomes). Another observation from this figure is that simulated jobs occupy a narrower diagonal than real jobs,

which spread on the entire width of the 25-75th percentile domain. This is because, as explained in Section V-B1, we do not have full information regarding the behavior of each individual job and make assumptions that make the job mix more homogeneous than in the real world. We could mitigate this by adding artificial noise to the models, but that would make the simulator non-deterministic, which is not desirable before reaching more confidence in the results.

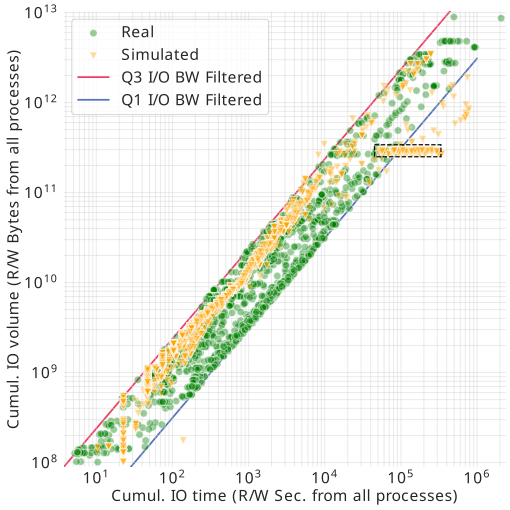


Fig. 7: Cumulative I/O volume vs. I/O time for real (green) and simulated (grey) regular jobs. Year of 2022, excl. training set (November).

Figure 8 is similar to Figure 6, but shows results for the whole year of 2022 and includes the three job classes ($\approx 1,250$ *slow* jobs in blue, $\approx 2,500$ *regular* jobs in grey and $\approx 1,250$ *fast* jobs in red). The pattern first observed in Figure 7 is once again shown inside a black-bordered box. This cluster causes the Pearson correlation on I/O time to drop to 0.43 for the *regular* jobs, while it would be 0.71 without it. An examination of the workload traces reveals that this cluster corresponds to a single job, which is executed more than 200 times and has a somewhat unique I/O behavior. The calibration procedure attempts to find a single configuration of the parameters that best fit all jobs on average, and the many invocations of this single job end up suffering from lower simulation accuracy. Because our simulator doesn't seem to be a good fit for this type of job, and it was repeated so many times over two months of our studied dataset, the impact is significant. Results also highlight a global high level of accuracy for *slow* jobs, with a Pearson correlation at 0.83. By contrast, *fast* jobs experience the worse correlation, at 0.52, although the overall trend is correct. This is because, as explained in Section V-B, to increase the scalability of FIVES we place artificial bounds on several parameters, such as the *stripe_count* and the number of files. This can lead to the execution of jobs with highly optimized I/O implementations to be simulated with lower levels of performance than in the real world. Note that our calibration of FIVES is for a particular trade-off between accuracy and scalability, which has allowed us to run large numbers of simulations as necessary for performing this research. For a given production use case, scalability could be reduced to increase accuracy, either overall or for particular job classes.

The results presented above are a first valuable step towards informing architecture and configuration of storage systems. However, FIVES's job-level accuracy is limited by the level of detail of the traces, which in turn limits the accuracy of a generic job model. Addressing this issue would require access to trace datasets with more complete per-job information, so that more detailed job models can be developed.

Challenge #2: Lesson learned

The high degree of I/O behavior heterogeneity among jobs makes it impossible to design a simulator that is accurate for all jobs. But it is possible to define job classes and achieve desirable trade-offs between simulation scalability and simulation accuracy for each class.

VIII. CONCLUSION

This work has presented the FIVES simulator of high-performance storage systems, which was developed using but also has contributed to state-of-the-art simulation frameworks. We have identified four challenges for the accurate simulation of high-performance storage systems, and have shown possible approaches for addressing these challenges. These include methods for modeling job I/Os, for coping with job heterogeneity, and for automating simulator calibration. An experimental evaluation based on one year's worth of supercomputer traces has allowed us to quantify achievable levels of simulation accuracy. Furthermore, we confirmed the coherence of our results with short experiments, including varying the number of OSTs in our platform model. This led to expected results in terms of bandwidth impact, conforming to our initial evaluation of FIVES. Although we have performed our investigation in the context of the Lustre file system, we believe the proposed methodologies and the FIVES simulator itself can be applied to a broad range of production settings.

The main direction for future work is the development of approaches for better handling the heterogeneity of job I/O behaviors. In this work we have resorted to defining job classes based on average I/O performance to handle this heterogeneity, which has proved effective but has limitations. We have defined our three jobs classes in a somewhat arbitrary manner, but it is likely that better classes could be defined. Developing a methodology to automatically pick the appropriate number of classes and the criteria for defining these classes would be a key advance. Regardless, within each job class there is still heterogeneity in the patterns of I/O operations (e.g., the number and frequency of distinct I/O phases throughout a job's execution). In this work we have not considered these patterns because the needed information was not present in our workload traces. As a result, within a job class, simulated job behaviors are artificially more homogeneous than in the

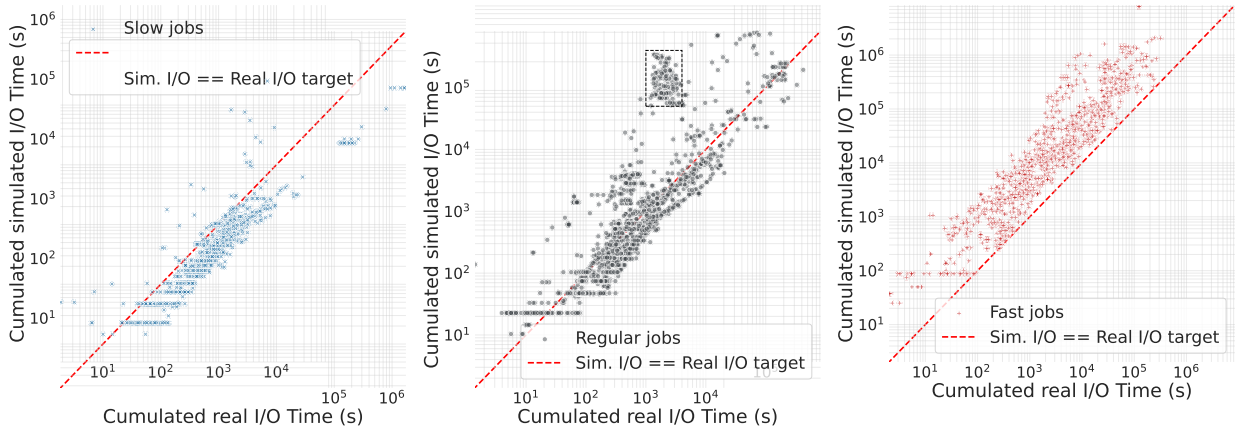


Fig. 8: Cumulative simulated I/O time vs. cumulative real I/O time - Full 2022 year, excluding training data of November (4,646 jobs) - From left to right, *slow*, *regular* and *fast* jobs.

real world. An important future research direction will be to augment FIVES so that it supports arbitrary I/O operation patterns and to calibrate it based on ground-truth data that does contain information regarding these patterns.

ACKNOWLEDGMENT

As part of the "France 2030" initiative, this work has benefited from a State grant managed by the French National Research Agency (Agence Nationale de la Recherche) attributed to the Exa-DoST project of the NumPEx PEPR program, reference: ANR-22-EXNU-0004. This research has also been supported in part by the NCSA-Inria-ANL-BSC-JSC-Riken-UTK Joint-Laboratory on Extreme Scale Computing (JLESC). The Darshan input data was generated from resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

REFERENCES

[1] "Top500 ranking," https://www.top500.org/.

[2] G. Lockwood, D. Hazen, Q. Koziol, R. Canon, K. Antypas, and J. Balewski, "Storage 2020: A Vision for the Future of HPC Storage," in *Report: LBNL-2001072*. Lawrence Berkeley National Laboratory, 2017. [Online]. Available: https://escholarship.org/uc/item/744479dp#author

[3] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor *et al.*, "Basic performance measurements of the intel optane dc persistent memory module," *arXiv preprint arXiv:1903.05714*, 2019.

[4] J. Lofstead, I. Jimenez, C. Maltzahn, Q. Koziol, J. Bent, and E. Barton, "Daos and friends: A proposal for an exascale storage system," in *SC '16: Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 585–596.

[5] J. Yang, J. Izraelevitz, and S. Swanson, "Orion: A distributed file system for {Non-Volatile} main memory and {RDMA-Capable} networks," in *17th USENIX Conf. on File and Storage Technologies (FAST 19)*, 2019, pp. 221–234.

[6] P. Schwan, "Lustre: Building a file system for 1,000-node clusters," in *Proc. of the Linux Symp.*, 2003, p. 9.

[7] M. Seiz, P. Offenhäuser, S. Andersson, J. Hötzer, H. Hierl, B. Nestler, and M. Resch, "Lustre i/o performance investigations on hazel hen: experiments and heuristics," *The J. of Supercomputing*, vol. 77, no. 11, p. 12508–12536, Nov. 2021.

[8] Z. Liu, R. Lewis, R. Kettimuthu, K. Harms, P. Carns, N. Rao, I. Foster, and M. E. Papka, "Characterization and identification of hpc applications at leadership computing facility," in *Proc. of the 34th ACM Int. Conf. on Supercomputing*. Barcelona Spain: ACM, Jun. 2020, p. 1–12. [Online]. Available: https://dl.acm.org/doi/10.1145/3392717.3392774

[9] S.-H. Lim, H. Sim, R. Gunasekaran, and S. S. Vazhkudai, "Scientific user behavior and data-sharing trends in a petascale file system," in *Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, Nov. 2017, p. 1–12. [Online]. Available: https://doi.org/10.1145/3126908.3126924

[10] J. McDonald, M. Horzela, F. Suter, and H. Casanova, "Automated Calibration of Parallel and Distributed Computing Simulators: A Case Study," in *Proc. of the 25th IEEE Int. Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC)*, 2024. [Online]. Available: https://arxiv.org/abs/2403.13918

[11] G. K. Lockwood, S. Snyder, T. Wang, S. Byna, P. Carns, and N. J. Wright, "A year in the life of a parallel file system," in *SC18: Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2018, pp. 931–943.

[12] A. K. Paul, O. Faaland, A. Moody, E. Gonsiorowski, K. Mohror, and A. R. Butt, "Understanding hpc application i/o behavior using system level statistics," in *2020 IEEE 27th Int. Conf. on High Performance Computing, Data, and Analytics (HiPC)*, 2020, pp. 202–211.

[13] J. L. Bez, A. M. Karimi, A. K. Paul, B. Xie, S. Byna, P. Carns, S. Oral, F. Wang, and J. Hanley, "Access patterns and performance behaviors of multi-layer supercomputer i/o subsystems under production load," in *Proc. of the 31st Int. Symp. on High-Performance Parallel and Distributed Computing*. Minneapolis MN USA: ACM, Jun. 2022, p. 43–55. [Online]. Available: https://dl.acm.org/doi/10.1145/3502181.3531461

[14] G. K. Lockwood, S. Snyder, S. Byna, P. Carns, and N. J. Wright, "Understanding data motion in the modern hpc data center," in *2019 IEEE/ACM Fourth Int. Parallel Data Systems Workshop (PDSW)*, Nov. 2019, p. 74–83.

[15] A. Khan, H. Sim, S. S. Vazhkudai, A. R. Butt, and Y. Kim, "An analysis of system balance and architectural trends based on top500 supercomputers," in *The Int. Conf. on High Performance Computing in Asia-Pacific Region*. Virtual Event Republic of Korea: ACM, Jan. 2021, p. 11–22. [Online]. Available: https://dl.acm.org/doi/10.1145/3432261.3432263

[16] F. Boito, G. Pallez, and L. Teylo, "The role of storage target allocation in applications' i/o performance with beegfs," in *2022 IEEE Int. Conf. on Cluster Computing (CLUSTER)*, 2022, pp. 267–277.

[17] F. Chowdhury, Y. Zhu, T. Heer, S. Paredes, A. Moody, R. Goldstone, K. Mohror, and W. Yu, "I/O Characterization and Performance Evaluation of BeeGFS for Deep Learning," in *Proc. Int. Conf. on Parallel Processing (ICPP)*. Association for Computing Machinery,

Aug. 2019, p. 1–10. [Online]. Available: https://dl.acm.org/doi/10.1145/3337821.3337902

[18] B. Behzad, S. Byna, Prabhat, and M. Snir, "Optimizing i/o performance of hpc applications with autotuning," *ACM Trans. on Parallel Computing*, vol. 5, no. 4, pp. 15:1–15:27, Mar. 2019.

[19] C. San-Lucas and C. L. Abad, "Towards a fast multi-tier storage system simulator," in *2016 IEEE Ecuador Technical Chapters Meeting (ETCM)*, 2016, pp. 1–5.

[20] H. Khetawat, C. Zimmer, F. Mueller, S. Atchley, S. S. Vazhkudai, and M. Mubarak, "Evaluating burst buffer placement in hpc systems," in *2019 IEEE Int. Conf. on Cluster Computing (CLUSTER)*, 2019, pp. 1–11.

[21] K. Arzymatov, M. Hushchyn, A. Sapronov, V. Belavin, L. Gremyachikh, M. Karpov, and A. Ustyuzhanin, "Online detection of failures generated by storage simulator," *J. of Physics: Conf. Series*, vol. 1740, no. 1, p. 012052, jan 2021. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1740/1/012052

[22] J. Monniot, F. Tessier, M. Robert, and G. Antoniu, "Supporting dynamic allocation of heterogeneous storage resources on hpc systems," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 28, p. e7890, 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.7890

[23] R. M. Fujimoto, "Parallel discrete event simulation," *Commun. ACM*, vol. 33, no. 10, p. 30–53, oct 1990.

[24] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross, "CODES: Enabling Co-Design of Multilayer Exascale Storage Architectures," in *Proc. of the Workshop on Emerging Supercomputing Technologies*, 2011.

[25] S. Böhm and C. Engelmann, "xSim: The extreme-scale simulator," in *Proc. of the Int. Conf. on High Performance Computing & Simulation*, 2011, pp. 280–286.

[26] M.-Y. Hsieh, R. Riesen, K. Thompson, W. Song, and A. Rodrigues, "SST: A Scalable Parallel Framework for Architecture-Level Performance, Power, Area and Thermal Simulation," *The Computer J.*, vol. 55, no. 2, pp. 181–191, 2012.

[27] "SimPy: Discrete event simulation for Python," https://simpy.readthedocs.io/en/latest/.

[28] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, Dec. 2002.

[29] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[30] S. Ostermann, R. Prodan, and T. Fahringer, "Dynamic Cloud Provisioning for Scientific Grid Workflows," in *Proc. of the 11th ACM/IEEE Int. Conf. on Grid Computing*, 2010, pp. 97–104.

[31] G. Kecskemeti, "DISSECT-CF: A simulator to foster energy-aware scheduling in infrastructure clouds," *Simulation Modelling Practice and Theory*, vol. 58, no. 2, pp. 188–218, 2015.

[32] E. U. Yousuf Khan, T. Rahim Soomro, and M. Nawaz Brohi, "iFogSim: A Tool for Simulating Cloud and Fog Applications," in *Proc. of the Int. Conf. on Cyber Resilience*, 2022, pp. 01–05.

[33] H. Casanova, R. Ferreira da Silva, R. Tanaka, S. Pandey, G. Jethwani, W. Koch, S. Albrecht, J. Oeth, and F. Suter, "Developing Accurate and Scalable Simulators of Production Workflow Management Systems with WRENCH," *Future Generation Computer Systems*, vol. 112, pp. 162–175, 2020.

[34] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *J. of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014. [Online]. Available: http://hal.inria.fr/hal-01017319

[35] P. Velho, L. Mello Schnorr, H. Casanova, and A. Legrand, "On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations," *ACM Trans. on Modeling and Computer Simulation*, vol. 23, no. 4, 2013.

[36] P. Velho and A. Legrand, "Accuracy Study and Improvement of Network Simulation in the SimGrid Framework," in *Proc. of the 2nd Intl. Conf. on Simulation Tools and Techniques*, 2009.

[37] A. Lebre, A. Legrand, F. Suter, and P. Veyre, "Adding storage simulation capacities to the simgrid toolkit: Concepts, models, and api," in *2015 15th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing*, 2015, pp. 251–260.

[38] A. Degomme, A. Legrand, G. Markomanolis, M. Quinson, M. Stillwell, and F. Suter, "Simulating MPI applications: the SMPI approach," *IEEE Trans. on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 2387–2400, 2017.

[39] A. Rizvi, T. Toha, M. Lunar, M. Adnan, and A. Alim Al Islam, "Cooling Energy Integration in SimGrid," in *Proc. of the 2017 Int. Conf. on Networking, Systems and Security (NSysS)*, 2017, pp. 132–137.

[40] L. Stanisic, S. Thibault, A. Legrand, B. Videau, and J.-F. Méhaut, "Faithful performance prediction of a dynamic task-based runtime system for heterogeneous multi-core architectures," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4075–4090, 2015.

[41] T. Cornebize, A. Legrand, and F. C. Heinrich, "Fast and Faithful Performance Prediction of MPI Applications: the HPL Case Study," in *Proc. of the 2019 IEEE Int. Conf. on Cluster Computing*, 2019, pp. 1–11.

[42] A. Mu'alem and D. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE Trans. on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, 2001.

[43] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale i/o workloads," in *2009 IEEE Int. Conf. on Cluster Computing and Workshops*, 2009, pp. 1–10.

[44] "Argonne national laboratory data catalog," https://reports.alcf.anl.gov/data/.

[45] T. I. Papon and M. Athanassoulis, "A Parametric I/O Model for Modern Storage Devices," in *Proc. of the 17th Int. Workshop on Data Management on New Hardware (DAMON)*. New York, NY, USA: Association for Computing Machinery, 2021.

[46] "Adaptive Experimentation Platform," https://ax.dev/, 2024.