ElastiSim: A Batch-System Simulator for Malleable Workloads

Taylan Özden
taylan.oezden@tu-darmstadt.de
Department of Computer Science
Technical University of Darmstadt
Darmstadt, Hesse, Germany

Tim Beringer
tim.beringer@tu-darmstadt.de
Department of Computer Science
Technical University of Darmstadt
Darmstadt, Hesse, Germany

Arya Mazaheri
arya.mazaheri@tu-darmstadt.de
Department of Computer Science
Technical University of Darmstadt
Darmstadt, Hesse, Germany

Hamid Mohammadi Fard
hamid.fard@tu-darmstadt.de
Department of Computer Science
Technical University of Darmstadt
Darmstadt, Hesse, Germany

Felix Wolf
felix.wolf@tu-darmstadt.de
Department of Computer Science
Technical University of Darmstadt
Darmstadt, Hesse, Germany

ABSTRACT

As high-performance computing infrastructures move towards exascale, the role of resource and job management systems is more critical now than ever. Simulating batch systems to improve scheduling algorithms and resource management efficiency is an indispensable option, as running large-scale experiments is expensive and time-consuming. Batch-system simulators are responsible for simulating the computing infrastructure and the types of jobs that constitute the workload. In contrast to rigid jobs, malleable jobs can dynamically reconfigure their resources during runtime. Although studies indicate that malleability can improve system performance, no simulator exists to investigate malleable scheduling policies. In this work, we present ElastiSim, a batch-system simulator supporting the combined scheduling of rigid and malleable jobs. To facilitate the simulation, we propose a malleable workload model and introduce a scheduling protocol that enables the evaluation of topology-, I/O-, and progress-aware scheduling algorithms. We validate the scaling behavior of our workload model by comparing training runtimes of various deep-learning models against the results achieved by ElastiSim. We use real-world cluster trace files to generate workloads and simulate various scheduling algorithms (FCFS, SJF, DRF, SRTF) to analyze their implications on the simulated platform. The results demonstrate that real-world executions show the same scaling behavior as our proposed workload model. We further show that ElastiSim can capture the complex interplay between emerging workloads and modern platforms to support algorithm designers by providing consistently meaningful results. ElastiSim is publicly available as an open-source project on https://github.com/elastisim.

CCS CONCEPTS

• **Software and its engineering** → **Scheduling**; • **Computing methodologies** → *Discrete-event simulation*; • **Computer systems organization** → Distributed architectures.

KEYWORDS

batch systems, simulations, malleable workloads, adaptive job scheduling, resource management

ACM Reference Format:
Taylan Özden, Tim Beringer, Arya Mazaheri, Hamid Mohammadi Fard, and Felix Wolf. 2022. ElastiSim: A Batch-System Simulator for Malleable Workloads. In *51st International Conference on Parallel Processing (ICPP '22), August 29-September 1, 2022, Bordeaux, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3545008.3545046

1 INTRODUCTION

Resource and job management systems (also referred to as *batch systems*, *job schedulers*, or *workload managers*) are essential to the efficient use of resources on large-scale and high-performance computing (HPC) systems. Conventionally, batch systems are responsible for scheduling jobs and providing resources in a highly contentious multi-tenant environment to maximize system efficiency and improve job completion times to satisfy user requirements. Batch systems apply scheduling algorithms that, depending on the objective, increase performance metrics such as system utilization, throughput, or energy efficiency.

However, the efficient use of resources depends not only on the applied scheduling algorithm but also on the types of jobs that constitute the workload. Although rigid jobs, which require a predefined and fixed number of resources, are still the most common job type in HPC environments, we observe an increasing interest in malleable jobs. In contrast to rigid jobs, malleable jobs can dynamically adapt to resource reconfigurations during runtime, providing schedulers with an additional instrument for optimization.[1] Given the rise of artificial intelligence and deep-learning (DL) workloads in HPC environments, malleability is increasingly becoming a subject of interest.

As large-scale experiments are expensive, time-consuming, and resource-intensive, simulations are indispensable for studying and improving job scheduling policies. Furthermore, continuous development and improvement of scheduling algorithms require extensive testing and debugging, which simulations can accelerate by multiple orders of magnitude. Thus, existing batch-system simulators such as Batsim [15] or Alea [32] allow users to integrate

[1]An alternative term is *elastic job*, which is predominant in the context of cloud computing.

custom scheduling algorithms to investigate their implications on varying workloads and platform architectures. However, to the best of our knowledge, no batch-system simulator supports the simulation of malleable workloads. Furthermore, the increasing interest in DL workloads, which are inherently malleable, require highly-customized scheduling algorithms that consider the interconnection of graphics processing units (GPUs) or parameters such as continuously updated progress of jobs, which existing simulators do not support.

To expedite the development of new scheduling policies for emerging workloads and address shortcomings of existing simulation methods, we present ElastiSim, a batch-system simulator supporting the combined scheduling of rigid and malleable jobs. First, we propose a workload modeling approach that allows users to describe malleable workloads efficiently using human-readable performance models. Second, we contribute a batch-system simulator, employing thoroughly validated network models provided by SimGrid [6], to evaluate multi-objective scheduling algorithms for malleable jobs based on our proposed workload model.

We demonstrate that our malleable workload model scales realistically by comparing simulations of DL workloads with experiments executed under various resource configurations on a high-performance computing (HPC) cluster. We further establish large-scale simulations and apply various scheduling algorithms to demonstrate the applicability of ElastiSim under malleable and rigid conditions.

Towards facilitating the simulation and evaluation of scheduling algorithms for malleable workloads, our main contributions are:

- A batch-system simulator supporting the combined scheduling of rigid and malleable jobs.
- A workload format to facilitate the description of malleable workloads using performance models.
- A simulation approach for custom schedulers enabling the evaluation of topology-, I/O-, and progress-aware scheduling algorithms.
- A SimGrid extension to simulate large-scale malleable GPU workloads.
- A detailed malleability study of DL training workloads.

The rest of this paper is structured as follows. In Section 2 we give insight into job types relevant to HPC environments and the attempts to support malleability in batch systems. Section 3 highlights notable work in the domain of batch-system simulations. In Section 4, we introduce our proposed workload model and how it facilitates describing malleable jobs and applications. Section 5 presents the architecture of ElastiSim, giving an overview of our simulation approach and underlying models. In Section 6, we describe our experimental evaluation approach, validate our workload model, and discuss the results achieved by ElastiSim in large-scale simulations. Finally, we conclude our contributions and discuss future work in Section 7.

2 BACKGROUND & MOTIVATION

To classify distinctive job characteristics, Feitelson and Rudolph [18] proposed four job categories: (1) *rigid jobs*, (2) *moldable jobs*, (3) *evolving jobs*, and (4) *malleable jobs*. According to this classification, rigid jobs are the most common job type and require a fixed number of resources throughout their execution. In the case of moldable jobs, the job scheduler, unlike rigid jobs, can assign an arbitrary number of resources before the job starts its execution. The remaining two job types can flexibly expand or shrink their resources during runtime. Evolving jobs initiate such resource reconfigurations on their own—provided that the system can conform to the request. In the case of malleable jobs, the scheduling system initiates resource reconfiguration, forcing the job to adapt to its newly assigned resources. Malleable and evolving jobs are often classified as *adaptive jobs*.

Malleable jobs can dynamically adapt to assigned resources and, therefore, have great potential to improve system performance. A well-known example of malleable workloads is the distributed training of deep neural networks (DNN). DNNs are usually trained using gradient descent algorithms that minimize a loss function and select appropriate weights for the connections within neural networks. Gradient descent algorithms are applied iteratively to a given dataset (i.e., epoch) during training to learn and minimize the error rate. Thus, the repetitive characteristic of DNN training allows schedulers to reconfigure the number of allocated resources at multiple points in time. Furthermore, the dataset is divided into equally sized mini-batches distributed among the allocated workers, increasing the workloads' scalability potentials and reconfiguration opportunities. Following the distribution of mini-batches, each worker computes the local gradient and then participates in averaging across all workers using all-reduce operations to obtain the global gradient. However, as not all DNN models are equally scalable and resource requirements often change during runtime, a workload-aware scheduler supporting malleable jobs is crucial to improve DNN training times.

Scheduling algorithms for adaptive jobs can enable batch systems to efficiently balance the overall system load and improve utilization, throughput, and job completion times [12, 14, 24, 42]. To facilitate dynamic resource allocation in real environments, researchers extended various workload managers to support malleable (and evolving) jobs [7–10, 42]. Furthermore, as batch systems require underlying runtime environments to support adaptation requests, solutions such as FlexMPI [39] or Invasive MPI (iMPI) [9] have extended the Message Passing Interface (MPI) [40] to support malleable jobs.

As we move towards exascale environments, batch systems must also consider a manifold of system resources in scheduling decisions to efficiently exploit the dynamic characteristics of malleable jobs. For instance, the increasing volume of I/O operations in HPC applications can lead to bottlenecks on traditional parallel file systems (PFSs) [54] which modern HPC environments prevent using high-performance storage systems such as burst buffers [38]. By considering such technologies, batch systems can apply I/O-aware scheduling algorithms to reduce PFS congestion and increase I/O throughput [23, 33]. In addition to advanced storage resources, HPC systems usually provide accelerators such as GPUs that, if required by the job, batch systems also have to account for the scheduling of those resources. Thus, evaluating novel scheduling algorithms has become crucial to expedite and facilitate the development of adaptive batch systems.

3 RELATED WORK

Numerous simulators to evaluate batch systems were developed in previous work, and most of them fall into one of the three following categories: (1) simulators built from scratch, (2) simulators based on a platform simulation framework, and (3) simulators extending an existing resource and job management system [32]. As the name suggests, simulators built from scratch do not base their simulations on existing frameworks. The second group of simulators employs platform simulation frameworks such as SimGrid or GridSim [3] and have the advantage of using validated simulation models. The last group of simulators directly extend real-world batch systems such as Slurm [55] and replace the actual job execution to advance the simulation clock.

Although no current simulator provides the infrastructure and the capability to simulate adaptive scheduling algorithms or malleable workloads, we outline the main characteristics of available batch-system simulators and highlight their differences in contrast to our approach. Driven by numerous studies, researchers developed and published many simulators to reinforce their scheduling approaches. Here, we focus on actively developed general-purpose simulators comparable to our approach.

A representative simulator built from scratch is AccaSim [21], a discrete-event simulator that divides job and resource management into two modules (*scheduler* and *allocator*) and provides several pre-defined resource managers. Users specify workloads in the standard workload format (SWF) [19], a standard for rigid workload models and real-world logs. In contrast to AccaSim, we focus on malleable workloads that the user can specify using our proposed application modeling approach. Similar to our approach, AccaSim provides system metrics to the scheduler during the simulation. While AccaSim focuses on power- and energy-aware and fault-resistant algorithms, we focus on I/O- and progress-awareness by periodically providing the I/O subsystem utilization and the progress of jobs to be considered in scheduling algorithms.

Several simulators employ frameworks that simulate the underlying computing infrastructure. The batch-system simulator Alea [32] follows that approach and is built on GridSim. Alea interprets SWF files and provides a pre-defined set of scheduling algorithms that users must extend at the source-code level. Platforms in Alea are specified using a custom format that comprises compute nodes, their number of CPUs, and available memory. In contrast, we focus on detailed platform descriptions extending SimGrid to model the complex behavior of computations, network communications, and advanced I/O concepts to allow the evaluation of topology-aware scheduling algorithms.

The batch-system simulator Batsim [15] follows an approach comparable to ElastiSim. It is based on SimGrid and allows developers to integrate algorithms in several languages based on inter-process communication. However, Batsim does not support malleable jobs and provides no possibility to specify performance models describing varying workloads. ElastiSim allows users to specify performance models that dynamically scale with the number of assigned resources at runtime. Furthermore, we enable large-scale malleable GPU simulations by providing a GPU model to investigate scheduling algorithms for workloads that are particularly predominant in fields such as deep learning.

Several approaches proposed and extended a Slurm-based simulator [30, 45, 50] that reads system logs to reproduce the workload and allows the evaluation of new algorithms with modifications to the codebase. The simulation engine replaces system calls to bypass actual job execution and accelerate the simulation. Although these simulators allow a seamless integration into production environments, they require detailed knowledge about the extended batch system. In contrast, our approach minimizes the effort to integrate new algorithms and does not rely on real-world logs or traces to simulate large-scale workloads.

4 WORKLOAD MODELING

The reliability of platform simulations depends highly on the workload executed in the simulated environment [17, 49]. To conduct representative simulations, users can rely on real-world traces and logs (often provided in standard formats such as SWF) or models that generate workloads derived from such logs, both available on public archives such as the *parallel workloads archive* [19]. Existing simulators, therefore, are either compatible with standard formats or propose a workload modeling approach for users to describe customized workloads. Custom modeling approaches usually describe workloads as a sequence of jobs with profiles representing the simulated application. These profiles can be as simple as application runtimes or more sophisticated, such as the number of floating-point operations or transferred bytes in network communications. As we tackle the simulation of malleable jobs, we face the necessity of workload models that consider application scalability and malleable characteristics, such as the overhead introduced by runtime reconfigurations.

To cover a broad range of applications, we studied workload characterizations on large-scale systems, indicating that applications often alternate between compute and I/O phases and that I/O requests are usually bursty [5, 31]. In addition, the rapid surge of interest in artificial intelligence and deep learning [29] over the last decade has driven technologies to catch up with the ever-increasing demand for GPU computations and communications. Considering the demands of modern workloads, we propose a workload modeling approach comprising jobs and application models. While jobs hold properties relevant to the scheduler, the application model reproduces the behavior of real-world applications in simulated environments. Our proposed application model comprises phases to rebuild the lifecycles of applications and tasks to cover typical activities of HPC applications. To address the requirements of malleable workloads, we employ performance models that scale with the number of resources and introduce runtime penalties on dynamic reconfigurations. Figure 1 portrays a high-level overview of our workload modeling approach, which we describe in the rest of this section.

4.1 Workload structure

We define a workload as a set of jobs arriving in the system, available for the scheduler to assign resources. ElastiSim supports rigid, moldable, and malleable jobs holding several properties, such as arrival time, the requested number of resources, or arguments used in performance models (see Section 4.2.2). In addition to properties, each job holds an underlying application model representing

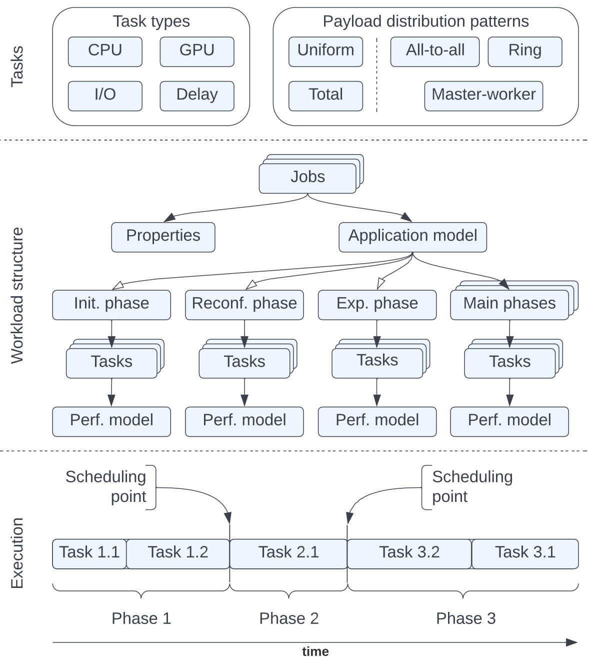

Figure 1: Our proposed workload modeling approach including the workload structure, available tasks, payload distribution patterns, and the execution flow. Filled arrow heads in the workload structure represent mandatory, and hollow arrow heads optional relationships.

the simulated application, which users can describe following a hierarchical building-block approach. Each application contains *phases* to model various stages within the application's lifecycle, and each phase is populated with *tasks*, representing simulated activities such as computations, network communication, or I/O operations. If specified, phases and tasks can further enable repetitions to facilitate designing repetitive workloads.

Considering malleable jobs, we require the application model to allow reconfiguration requests of the scheduler at specific points during runtime. Thus, we introduce *scheduling points* that are placed between all phases—and at the end of all phase repetitions—of malleable jobs by default. Scheduling points are locations where an application can expand or shrink its resources if the scheduler requests a reconfiguration.

However, runtime reconfigurations can create additional overhead as the application adapts to its new configuration. Possible sources for introduced overhead can be newly assigned compute nodes required to fetch application data from the PFS or data redistribution [39]. We define that overhead as *reconfiguration penalty* and allow users to define an optional *reconfiguration* and *expansion* phase. The reconfiguration phase specifies tasks executed on all resources after a reconfiguration occurs. On the other hand, the expansion phase specifies tasks executed only on newly allocated resources (i.e., expanded resources). We further provide a particular *initialization* phase to specify tasks executed only on the initial configuration of a job.

4.2 Tasks & payloads

Tasks in ElastiSim represent the activities that generate the actual load on the simulated platform, targeting various resources. To specify resource utilization, we introduce task *payloads* describing the amount of work that has to be processed, such as floating-point operations (FLOPs) or the number of bytes to communicate.[2] We distinguish three types of tasks, each carrying a different type of payload: (1) compute, (2) I/O, and (3) generic delays. Compute tasks are specified using the number of FLOPs to process and can target either CPUs or GPUs. To simulate I/O operations, users specify the number of bytes to read or write and choose a storage system (PFS or burst buffer). Delay tasks are generic time-based activities specified by the duration to occupy assigned resources. To increase the flexibility of our application model, we further introduce sequence tasks, which are generic containers that can contain any task—sequences included. Sequence tasks allow users to model irregularities in simulated applications, such as checkpointing to the PFS by combining various tasks in a repeated sequence before the checkpoint takes place.

Although defining explicit communication tasks are optional, compute tasks implicitly impose communication payloads. Such a paradigm has two main advantages: (1) it allows users to overlap computation and communication, and (2) depending on the resource (CPU or GPU), the communication takes place using different network links. In the case of GPU tasks, the simulator considers communication of GPU devices on the same host as intra-node communication (e.g., NVLink) and with devices on other hosts as inter-node communication (e.g., InfiniBand).

4.2.1 Payload distribution patterns.

Payload distribution patterns. To define the distribution of payloads among resources, we introduce *payload distribution patterns*. We distinguish two types of patterns; regular and communication patterns. While regular patterns define payload distribution for compute, I/O, and delay tasks, communication patterns define payload distribution in communication tasks. We support two regular patterns that define whether each allocated resource processes the specified payload (uniform) or whether the payload is distributed evenly (total). On the other hand, communication patterns construct communication matrices specifying the number and direction of exchanged bytes among resources. ElastiSim currently supports all-to-all, ring, and master-worker patterns to cover common communication scenarios in HPC applications.

In the case of rigid jobs, we support an additional method of defining payloads. As the number of resources is known prior to execution, users can manually define resource utilization using a vector or a matrix to describe task payloads that are not generated based on patterns.

4.2.2 Performance Models.

Performance Models. To allow dynamic payload adjustments for malleable workloads, we use performance models that scale with the number of assigned resources. Performance models are

[2]Although payloads are commonly used in telecommunication to describe the content of a message, we use the term payload as the load a task carries independent of its type.

human-readable mathematical models that characterize application behavior under various resource configurations. These models describe metrics such as the number of FLOPs, the number of bytes sent through the network, or simply the application runtime. Users can describe performance models by inspecting applications, or using tools such as Extra-P [4] to facilitate creating models by empirically analyzing the application under multiple configurations. Performance models in ElastiSim can depend on the number of assigned resources (compute nodes or GPUs) and arguments specified in job properties. The simulation runtime evaluates performance models to update the payloads at every task (re)configuration utilizing the mathematical expression library ExprTk [41]. In addition to various mathematical functions and operations, ExprTk supports control structures such as ternary conditions or string operations, allowing users to describe varying workloads using the same application and performance models dependent on the provided job arguments.

5 ELASTISIM ARCHITECTURE

According to the classification introduced in Section 3, ElastiSim is classified as a simulator based on a platform simulation framework. Such frameworks facilitate the simulation of computing infrastructures and expose an API to integrate the framework into higher-level software.

State-of-the-art platform simulation frameworks are discrete-event simulators that model networks either using a *packet-level* or *flow-level* approach. Packet-level simulators such as CODES [11] model every network activity as a particular event in the simulation. Given the excessively high number of packets routed through the network in large-scale computing infrastructures, packet-level simulators can be extraordinarily resource-intensive and often require distributed memory architectures. Flow-level simulators such as SimGrid [6] define network communication as data flows consuming the available bandwidth on network links. Each link has a predefined capacity (i.e., bandwidth), and each flow has a specific rate describing the bandwidth consumption. The sum of all flow rates on a link must not exceed its predefined capacity. Although recent packet-level approaches present promising results [1], flow-level simulations remain significantly faster than consolidated packet-level simulators, as they do not model every network packet as an individual event [16, 20].

Even though packet-level simulators are closer to real packet-switched networks, we focus on fast but reliable simulations of entire platforms (including computational resources) rather than a fine-grained analysis of the network and the effects of various packet routing mechanisms. We, therefore, establish batch-system and platform simulations using a flow-level backend to enable fast and scalable simulations on widely available commodity hardware such as laptops. SimGrid's flow-level approach is highly scalable [2, 43], validated [53], and used in hundreds of publications [27] making it the ideal candidate to provide platform simulations for ElastiSim.

SimGrid supports various distributed computing systems (e.g., HPCs, clouds, and grids) primarily composed of hosts, network links, and routes describing the underlying network infrastructure. Computing systems in SimGrid are described by platform files that

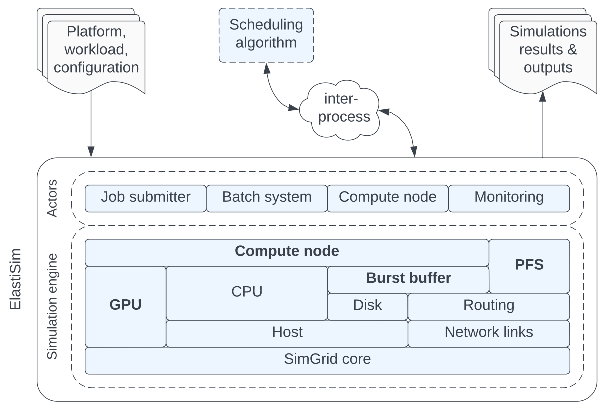

Figure 2: The architecture of ElastiSim, including the simulation engine, system actors, the external scheduling algorithm, and input and output files. Our extensions to the simulation engine are highlighted in bold text. Compute nodes are part of the simulation engine *and* system actors as they are considered resources but also continuously interact with the batch system.

either specify predefined topologies (e.g., dragonfly, fat tree, or torus) or describe individual hosts and their network interconnect. SimGrid hosts represent computing units defined by their computational power (i.e., floating-point operations per second) and additional resources such as hard disks. To establish communication, hosts utilize network links that can be assigned to multiple routes, simulating shared bandwidth on the underlying topology.

However, SimGrid is not a simulator but a versatile simulation framework building the foundation for other simulators to investigate platform utilization in various scenarios. Thus, we extend SimGrid to establish malleable batch system simulations and describe our proposed system architecture, models, and actors in the rest of this section.

5.1 System overview

ElastiSim is a simulator written in C++ that employs and extends SimGrid features to establish the simulation of an entire batch system and all interacting actors. We introduce resource management based on the allocation of compute nodes, extend SimGrid to support GPUs, introduce PFS and burst buffer semantics, and model every interaction relevant to workload management in large-scale distributed computing infrastructures. We further establish interfaces for users to integrate their scheduling algorithms using inter-process communication and propose a scheduling protocol that allows the evaluation of adaptive scheduling algorithms for malleable workloads. As illustrated in Figure 2, we propose an architecture that separates the concerns of platform simulations, the management of events initiated by system actors, and the scheduling algorithm provided by the user.

To run simulations, users have to provide multiple inputs. The scheduling algorithm is the essential input that receives information from the batch system and forwards scheduling decisions.

Scheduling algorithms are separate processes running individually outside ElastiSim, establishing the connection using the networking library ZeroMQ [26]. This design allows users to integrate interfaces and develop algorithms in any language supported by ZeroMQ. We support and provide a Python interface in the current version of ElastiSim. Using the proposed scheduling protocol (see Section 5.5), users can forward schedule decisions to ElastiSim. To describe the simulated platform, we rely on SimGrid platform descriptions, which the user extends with ElastiSim-specific properties. We use these properties to introduce advanced semantics for storage systems (see Section 5.2) or extended features such as GPUs (Section 5.3). To simulate different scenarios, users provide the workload model consisting of jobs and application models as described in Section 4. Lastly, by providing a configuration file, users can define simulation parameters such as the scheduling or monitoring interval. ElastiSim provides detailed outputs and visualizations, including job- and schedule-centric results and resource utilization as comma-separated value (CSV) files.

5.2 Storage model

ElastiSim provides the semantics for two types of storage systems, parallel file systems, and node-local burst buffers. Here, we describe our simulation approach for both storage systems.

5.2.1 Parallel file system. We provide semantics for parallel file systems (PFSs) by modeling them as dedicated nodes representing a storage system with distributed access behind a single namespace that users can indicate in the provided platform file. Our approach allows the PFS to be accessed via numerous routes, effectively increasing the bandwidth if accessed in parallel. To limit the bandwidth of the underlying file system, users attach a network link at the last level to specify the maximum available bandwidth. Using asymmetric routes, the PFS can provide independent read and write bandwidths. Such a simple yet effective model allows us to study the effects of data transfer in networks and identify bottlenecks during data-intensive workload phases.

5.2.2 Burst buffer. Modern HPC environments include an additional high-performance storage layer between compute nodes and the PFS—the *burst buffer* (BB). Two representative architectures are predominant in HPC environments: *node-local* and *remote-shared* BBs. Node-local BB architectures provide high-bandwidth storage on compute nodes, while remote-shared BBs are dedicated high-performance storage systems located in the network accessible by all compute nodes through a global namespace. ElastiSim supports node-local BBs that users can specify in the platform description to generate BBs on compute nodes. Each burst buffer task specified in the application model targets the local BBs of assigned compute nodes. Although developers can accelerate applications using node-local BBs, compute nodes can only access their local BBs, narrowing the use cases such architectures can support. To enable data sharing and collective access on node-local BBs, file systems such as BeeOND [51] or GekkoFS [52] introduce *wide striping*. Wide-striped file systems combine the storage space of all node-local disks assigned to a particular job and provide access behind a single namespace. ElastiSim supports both node-local BB approaches, exclusive and wide-striped access. In the case of wide-striped file systems, ElastiSim automatically introduces network communication for each BB task, representing striped access to local disks of distinct compute nodes.

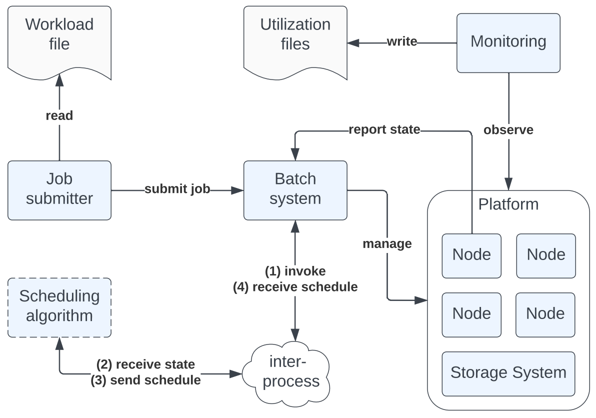

Figure 3: Actors and their responsibilities in ElastiSim. The scheduling algorithm is not considered a separate actor and runs outside ElastiSim as a stand-alone process represented by dashed lines.

5.3 GPU model

Our GPU model considers multiple GPUs per compute node, connected using dedicated links. As SimGrid does not support GPUs, we provide them as separate computational resources utilized during task execution. Users specify the number of GPUs per node, their computational performance, and the bandwidth of each connecting GPU link. As compute nodes are the primary resources assigned to jobs, we distribute GPU computations among the requested GPUs located on the assigned compute node. Communications among GPUs on a compute node utilize the specified GPU links (intra-node communication). In contrast, communications among GPUs on distinct nodes utilize the corresponding network links of the compute node, creating traffic on the underlying platform topology (inter-node communication).

5.4 System actors

ElastiSim is a discrete-event simulator modeling all components of the simulated system as separated modules. To integrate individual simulation logic and generate activities on the platform, we introduce actors for all modules to simulate their corresponding roles, such as the job submitter or the batch system. All actors run concurrently and exchange information when relevant events occur (e.g., job submission). As shown in Figure 3, ElastiSim is composed of three interacting (job submitter, batch system, and compute nodes) and one observing actor (monitoring).

The first actor initiating events in the simulation is the job submitter, reading the provided workload file and initiating events to inform the batch system of incoming jobs at the specified submission times.

The batch system is the central unit representing the bridging element between the user and the platform, including compute nodes. It queues jobs submitted by the user and is responsible for periodically invoking the scheduling algorithm. Although the scheduling algorithm lives outside the simulator, the batch system considers the algorithm as an integral component of itself. The batch system is further responsible for resource management, allocating compute nodes for jobs specified in the schedule decision.

We define *compute nodes* as the primary resources managed by the batch system. Compute nodes are considered resources but, at the same time, actors as well. They are permanently in contact with the batch system, forwarding the progress of executed jobs and communicating scheduling points and job completions. Compute nodes receive allocation requests from the batch system and are the only components performing activities that create load on the platform, which is observable by the monitoring module.

The monitoring module periodically gathers performance metrics to provide detailed simulation outputs. The observed metrics include CPU and GPU utilization per node, the overall network activity, and the load on the I/O subsystem.

5.5 Scheduling protocol

The scheduling protocol defines the communication, and the exchanged information between the simulated batch system, the scheduling algorithm, and compute nodes. The batch system invokes the algorithm periodically in a user-defined interval to allow reconfigurations of malleable jobs during runtime. Users might also specify flags to invoke the scheduling algorithm on job submission and completion. We allow users to specify minimum scheduling intervals to prevent multiple invocations during short periods (e.g., a high number of job submissions within a short period).

Each invocation contains the following information gathered at the time of invocation: (1) the job queue, (2) the state of each compute node, and (3) the utilization of the I/O subsystem. We include running jobs in the job queue that report their progress to facilitate the development of progress-aware scheduling algorithms. The progress is defined by the number of completed and the total number of phases (excluding initialization, reconfiguration, or expansion phases). The reported state of each compute node includes assigned jobs and GPU allocations. Lastly, the invocation includes the monitored utilization of the I/O subsystem to identify potential bottlenecks and enable I/O-aware schedules.

The scheduling algorithm is responsible for assigning compute nodes to jobs. Each node can be assigned to a job individually, enabling topology-aware schedules (e.g., assigning neighboring nodes to the same job). ElastiSim also supports the oversubscription of compute nodes, in which case the jobs running on one compute node share its resources. After making the schedule decision, the algorithm returns the list of node assignments for all configured and reconfigured jobs. In the case of initial job configurations, the batch system applies scheduling decisions immediately and allocates the corresponding nodes. Reconfigurations are stored separately and applied when the job reaches its next scheduling point. The batch system updates the state of malleable jobs that have a pending reconfiguration to allow the scheduling algorithm to (or not to) consider them in upcoming iterations. In addition to job configurations, the scheduling algorithm can flag jobs for termination. The batch system applies job terminations immediately.

6 EXPERIMENTAL RESULTS

To evaluate our proposed simulator, we established two experiments to (1) validate our workload and application model and (2) assess the applicability of our simulator by comparing various scheduling policies under malleable and rigid conditions. We built a foundation for our experiments to represent popular HPC workloads by training various deep-learning (DL) models as a test case for malleable applications using multiple configurations on a GPU cluster. To validate the correctness of our workload and application model, we rebuilt the utilized GPU cluster in ElastiSim, described the DL models as malleable applications that scale with the number of assigned GPUs, and compared the runtimes with the simulation results. We further conducted large-scale workload simulations derived from trace files from a state-of-the-art DL cluster and applied multiple scheduling algorithms to gain insight into the achieved performances by each algorithm.

6.1 Experimental setup

Testbed. To validate our workload and application model, we trained various convolutional neural networks (CNNs) on a GPU cluster with different configurations ranging from one to eight compute nodes, with each node containing four NVIDIA Tesla V100 16GB GPUs that were all allocated during the experiments. We trained the CNNs using the distributed DL training framework Horovod [47] 0.24.2 with PyTorch 1.11, and CUDA 11.5 with models from torchvision 0.12. We measured the execution time of the forward pass, backward pass, gradient update, and all-reduce operation using NVIDIA Nsight 2022.1.1. Moreover, all ElastiSim simulations were performed on an AMD Ryzen 7 PRO 4750U mobile CPU.

Workload setting. We randomly chose 400 jobs from traces of the Microsoft DL cluster Philly [28] to simulate various DL workloads. We used the submission times provided in the trace file. However, the trace file does not include any information about the DL model and dataset trained in each job. Therefore, we randomly assigned DL models to each job that we have modeled as an application model for the simulation. Furthermore, we set a random epoch size for each job, ranging from 50 to 100. We applied the evaluated scheduling algorithms to the same workload to guarantee a fair comparison.

Simulated platform. For our simulations, we rebuilt the GPU cluster on which we conducted our experiments in ElastiSim. The GPU cluster is organized in a fat-tree topology and has a dedicated GPU partition on which each node contains four GPUs. The GPUs are fully connected using duplex links, each with a bandwidth of 25 GB/s (intra-node connection). Compute nodes are connected using two network interface controllers (NICs), each providing a bandwidth of 100 Gbit/s (inter-node connection). For our large-scale simulations of various scheduling algorithms, we modeled a dragonfly topology with 192 nodes, each equipped with four GPUs. We increased the intra-node connection of GPUs to 50 GB/s per

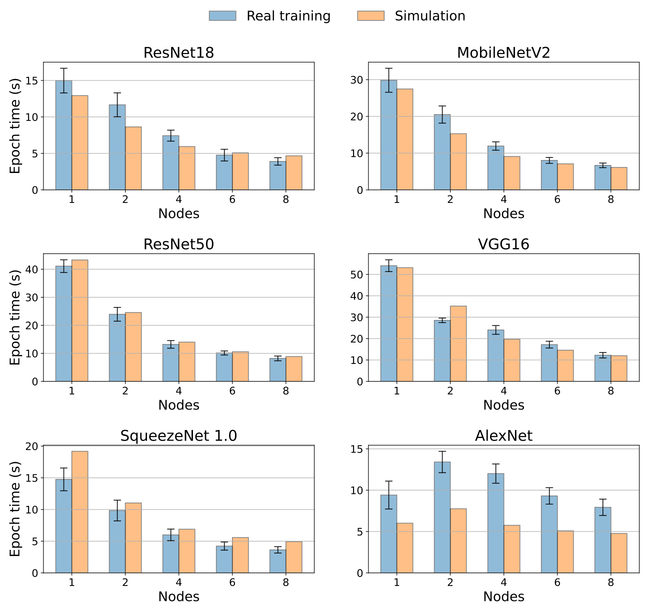

Figure 4: Training times per epoch of real and simulated training.

link and further equipped each node with four NICs with a total bandwidth of 800 Gbit/s (200 Gbit/s per link).

6.2 Deep-learning application model

As described in Section 2, DL is a popular workload providing malleable characteristics through its inherently repetitive pattern. We validate our workload and application model by replicating the distributed training of convolutional neural networks (CNNs) using the approach described in Section 4. Furthermore, as ElastiSim supports job arguments, we simulate various CNNs and datasets without modifying the underlying application model. Thus, we pass the number of activations and parameters of the DL model in addition to the size of the dataset as job arguments. Using these properties has a significant advantage in that we are not required to perform any real training or runtime measurements to describe the scalability of DL applications.

Defined by our application model, each compute node initially reads the training and validation dataset from the PFS. If the scheduler assigns additional nodes during runtime, each newly allocated node must perform the same task as a reconfiguration penalty before taking part in the simulated DL training. We model each epoch as a recurrent phase, implicitly creating scheduling points allowing the scheduler to resize the resources after each epoch. We further simulate a training step for each batch on a GPU as a sequence of tasks, repeated for the number of batches divided by the number of allocated GPUs.

Every DL training step consists of three sub-tasks. We choose to model these tasks as they are compute- and communication-intensive and contribute directly to the overall job runtime.

(1) **Forward/backward pass:** We simulate the forward and backward pass of the DL model with one batch of data by two sequential delay tasks. As the number of activations correlates highly with the inference time [44], we estimate the duration of these two tasks by a function (i.e., performance model) that scales with the number of activations of the DL model.
(2) **Gradient synchronization:** The synchronization of gradient updates among GPUs are simulated using a ring all-reduce communication pattern, which is commonly used for this objective in frameworks such as Horovod. The payload of this task is the number of the parameters in the DL model, split into $2 \times (N - 1)$ communication steps, where N is the number of allocated GPUs.
(3) **Gradient update:** The synchronized gradient updates are applied to the local DL models on each GPU. We estimate the duration of this step with a function that scales with the number of parameters in the DL model.

At the end of each epoch, a final computational task that scales equally to the forward pass simulates the validation step of the DL training. Additionally, the trained DL model may checkpoint to the PFS after a specified number of epochs, provided as an argument in the job specification.

Runtime analysis. We measured the epoch runtimes for MobileNetV2 [46], ResNet18-50 [22], VGG16 [48], Alexnet [35], and SqueezeNet [25] with the training dataset from CIFAR-10 [34], containing 50000 images upscaled to 224x224 pixels and the batch size of 64. We then simulated the training of the same DL models and dataset using the described application model to compare the measured runtimes with the simulated runtimes achieved by ElastiSim.

Figure 4 presents the duration of an epoch for the real and simulated training for different numbers of nodes. Our results show that our simulated application model has similar runtimes to MobileNetV2, ResNet18, ResNet50, and VGG16, with a relative error of 5.3–17.2%. However, for AlexNet and SqueezeNet, we notice that relative errors increase compared to real runtimes. We observed that such a higher error rate is due to our performance model that cannot fully capture the runtime of the gradient update within the DL training. Furthermore, the relative error of the forward and backward passes is higher if they are less computationally intensive. We believe that using larger models and batch sizes would address this issue. Nonetheless, we can observe that the simulated scaling behavior is preserved for all node configurations, which is particularly evident in the case of AlexNet, which yields higher epoch runtimes on two allocated nodes (i.e., eight GPUs) than on one allocated node. Although the relative error in runtime for AlexNet is higher than for the other simulated DL models, this particular behavior is reproduced in the simulated results. Considering the intra- and inter-node communication between GPUs and compute nodes, our results indicate that our simulation approach can capture the behavior of such complex workloads.

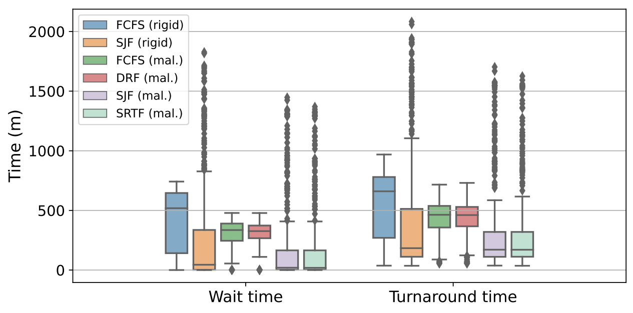

Figure 5: Wait and turnaround times for the evaluated scheduling policies.

6.3 Scheduling algorithms

We simulated the training of various DL models on a cluster with 192 (i.e., 768 GPUs) compute nodes using two rigid and four malleable scheduling policies. The simulated rigid scheduling algorithms comprise first come first serve (FCFS) and shortest job first (SJF). While we extend FCFS and SJF to support malleability, we provide two additional scheduling algorithms for malleable jobs: dominant resource first (DRF) and shortest remaining time first (SRTF).

FCFS processes jobs in the order of their arrival time. To increase system utilization, we extend the rigid FCFS algorithm to backfill jobs without preventing delays of jobs that have arrived earlier. SJF estimates the expected job duration by combining epoch size and the iteration time, approximated using the performance model of the backward pass. We use the backward pass because it correlates with the iteration time, making it suitable to estimate the remaining time compared to other jobs, depending on the trained model. The malleable version of SJF uses the same estimation to expand jobs during runtime. DRF prioritizes jobs for resource assignment based on their dominant resource share (i.e., the number of GPUs). SRTF is a progress-aware algorithm estimating the remaining time using the job progress information provided by ElastiSim.

Figure 5 presents the wait times in the queue and the turnaround times spanning from job submission to completion as box plots. Rigid FCFS results in the highest wait and turnaround times as it sequentially schedules incoming jobs, except for a small number of backfilled jobs. Furthermore, we can observe that the malleable counterparts of FCFS and SJF perform better than the rigid version. Malleable SJF and SRTF deliver the best performance among all scheduling algorithms, as they both consider the duration of jobs or even the progress, as in the case of SRTF. However, the outliers in wait and turnaround times of the SJF and SRTF algorithms indicate that they favor shorter jobs, leading to some jobs that are de-prioritized and remain in the queue for longer durations.

In Figure 6, we show the waiting and running times of the jobs applying the evaluated scheduling algorithms. The batch system sequentially assigns job IDs based on the submission time. We observe that malleability decreases scheduling makespans compared to their rigid counterparts (21 % for FCFS, 18 % for SJF). Furthermore, we can immediately identify the characteristics of the FCFS algorithm allocating resources to jobs consecutively based on their order of arrival. As the DRF algorithm initially assigns the minimum number of resources to jobs, we can identify a stepped pattern indicating that multiple jobs start simultaneously. For SRTF, the results show that shorter jobs are scheduled earlier, and longer jobs remain longer in the queue, confirming the observation of outliers in Figure 5.

The results demonstrate that ElastiSim provides a robust simulation infrastructure for scheduling malleable and rigid workloads. The evaluated outputs give valuable job- and schedule-centric insights and allow algorithm designers and engineers to develop novel scheduling approaches that consider the interplay of emerging workloads and the underlying platform architecture.

7 CONCLUSION & OUTLOOK

In this paper, we presented ElastiSim, a batch-system simulator supporting the combined scheduling of rigid and malleable jobs. We proposed a workload modeling approach based on performance models and established batch-system simulations compatible with platforms that introduce advanced storage system semantics and enable large-scale GPU simulations. Our proposed scheduling approach facilitates the evaluation of novel schedulers that are required to consider a manifold of system resources and deal with the intricacies of emerging workloads. Furthermore, we conducted experiments to validate the correctness of our workload and application model and performed simulations of large-scale workloads derived from trace files of a large-scale DL cluster. Our results demonstrate that our proposed workload model represents the behavior of real-world workloads, and ElastiSim provides meaningful results under malleable and rigid conditions.

For future versions of ElastiSim, we plan several updates targeting the underlying simulation architecture and scheduling semantics. The first update will introduce remote-shared burst buffer support to simulate and cover more data-intensive workload scenarios. Established solutions (e.g., Cray's DataWarp [36] or DDN's Infinite Memory Engine [13]) introduce advanced semantics such as transparent caching [37], automatically migrating data to the PFS, which we plan to support by simulating the implied behavior and its effects on the platform. Additionally, we plan to provide advanced semantics for workflows to support dependencies among jobs in ElastiSim. Lastly, as we already support scalable workloads, we plan to extend our application model with reconfiguration requests initiated by the application to support evolving jobs.

ACKNOWLEDGMENTS

We are thankful to Marc-André Vef for his insightful feedback on storage systems in HPC environments. We acknowledge the support of the European Commission and the German Federal Ministry of Education and Research (BMBF) under the EuroHPC Programme ADMIRE (GA No. 956748, BMBF funding No. 16HPC006K). The EuroHPC Joint Undertaking (JU) receives support from the European Union's Horizon 2020 research and innovation programme and GER, FRA, ESP, ITA, POL & SWE. This research was also supported by the EBRAINS research infrastructure, funded by the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific GA No. 945539 (Human Brain Project SGA3), and is partly funded by the Federal Ministry of Education and Research (BMBF) and the state of Hesse as part of the NHR and

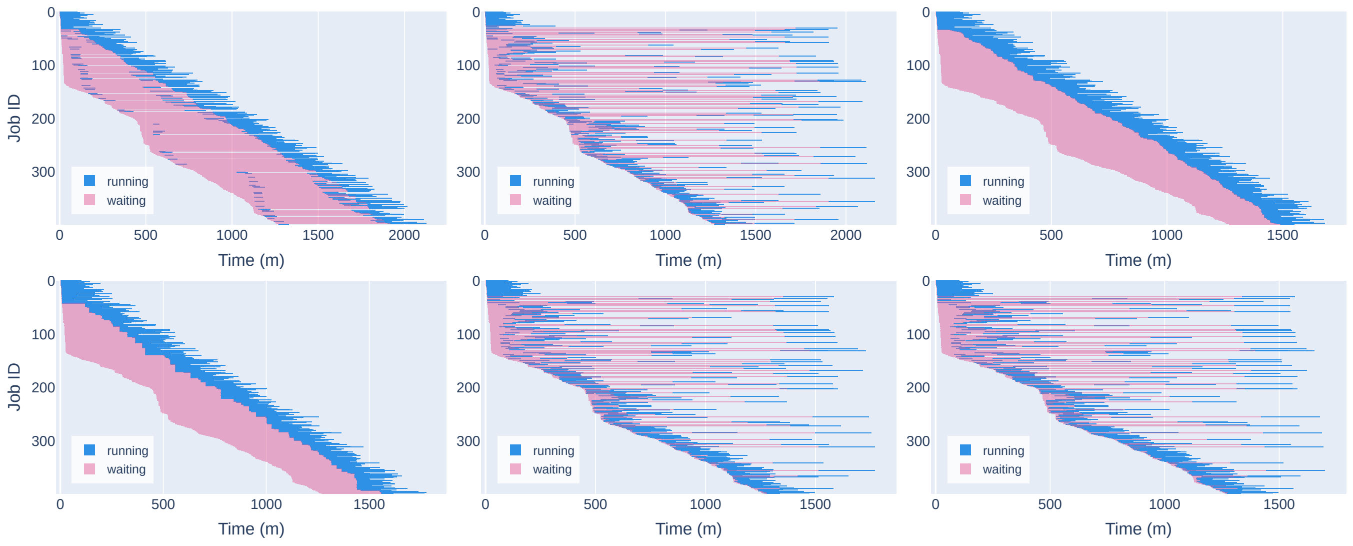

Figure 6: Wait and running times of jobs for rigid FCFS (upper left), rigid SJF (upper center), malleable FCFS (upper right), malleable DRF (lower left), malleable SJF (lower center), and malleable SRTF (lower right).

Software Campus programs. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS at Jülich Supercomputing Centre (JSC).

REFERENCES

[1] Maciej Besta, Marcel Schneider, Salvatore Di Girolamo, Ankit Singla, and Torsten Hoefler. 2021. Towards Million-Server Network Simulations on Just a Laptop. arXiv:2105.12663

[2] Laurent Bobelin, Arnaud Legrand, Márquez Alejandro González David, Pierre Navarro, Martin Quinson, Frédéric Suter, and Christophe Thiery. 2012. Scalable Multi-Purpose Network Representation for Large Scale Distributed System Simulation. In *Proc. of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 19.

[3] Rajkumar Buyya and Manzur Murshed. 2002. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience* 14, 13-15 (2002), 1175–1220.

[4] Alexandru Calotoiu, Torsten Hoefler, Marius Poke, and Felix Wolf. 2013. Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes. In *Proc. of the ACM/IEEE Conference on Supercomputing (SC13)*. ACM, 1–12.

[5] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011. Understanding and Improving Computational Science Storage Access through Continuous Characterization. *ACM Trans. Storage* 7, 3, Article 8 (2011).

[6] Henri Casanova, Arnaud Legrand, and Martin Quinson. 2008. SimGrid: A Generic Framework for Large-Scale Distributed Experiments. In *Proc. 10th International Conference on Computer Modeling and Simulation (uksim 2008) (ICCMS)*. 126–131.

[7] Marcia C. Cera, Yiannis Georgiou, Olivier Richard, Nicolas Maillard, and Philippe Olivier Alexandre Navaux. 2009. Supporting MPI Malleable Applications upon the OAR Resource Manager. In *Colloque d'Informatique: Brésil / INRIA, Coopérations, Avancées et Défis (COLIBRI)*.

[8] Mohak Chadha, Jophin John, and Michael Gerndt. 2020. Extending SLURM for Dynamic Resource-Aware Adaptive Batch Scheduling. In *Proc. of the IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. 223–232.

[9] Isaías Comprés, Ao Mo-Hellenbrand, Michael Gerndt, and Hans-Joachim Bungartz. 2016. Infrastructure and API Extensions for Elastic Execution of MPI Applications. In *Proc. of the 23rd European MPI Users' Group Meeting (EuroMPI)*. ACM, 82–97.

[10] Isaías A. Comprés Ureña and Michael Gerndt. 2019. Towards Elastic Resource Management. In *Proc. of the 11th International Workshop on Parallel Tools for High Performance Computing (PTHPC)*, Christoph Niethammer, Michael M. Resch, Wolfgang E. Nagel, Holger Brunst, and Hartmut Mix (Eds.). Springer, 105–127.

[11] Jason Cope, Ning Liu, Sam Lang, Phil Carns, Chris Carothers, and Robert Ross. 2011. Codes: Enabling co-design of multilayer exascale storage architectures. In *Proc. of the Workshop on Emerging Supercomputing Technologies (WEST, Vol. 2011)*. ACM.

[12] Marco D'Amico, Ana Jokanovic, and Julita Corbalan. 2019. Holistic Slowdown Driven Scheduling and Resource Management for Malleable Jobs. In *Proc. of the 48th International Conference on Parallel Processing (ICPP)*. ACM.

[13] DDN. 2022. IME®FLASH-NATIVE DATA CACHE | DDN. https://www.ddn.com/products/ime-flash-native-data-cache/.

[14] Briag Dupont, Nesryne Mejri, and Georges Da Costa. 2020. Energy-aware scheduling of malleable HPC applications using a Particle Swarm optimised greedy algorithm. *Sustainable Computing: Informatics and Systems* 28 (2020).

[15] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. 2016. Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator. In *Proc. 20th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*.

[16] Kolja Eger, Tobias Hoßfeld, Andreas Binzenhöfer, and Gerald Kunzmann. 2007. Efficient Simulation of Large-Scale P2p Networks: Packet-Level vs. Flow-Level Simulations. In *Proc. of the 2nd Workshop on Use of P2P, GRID and Agents for the Development of Content Networks (UPGRADE)*. ACM, 9–16.

[17] D.G. Feitelson. 2003. Metric and workload effects on computer systems evaluation. *Computer* 36, 9 (2003), 18–25.

[18] Dror G. Feitelson and Larry Rudolph. 1996. Towards Convergence in Job Schedulers for Parallel Supercomputers. In *Proc. of the 2nd Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. Springer, 1–26.

[19] Dror G. Feitelson, Dan Tsafrir, and David Krakov. 2014. Experience with using the Parallel Workloads Archive. *J. Parallel and Distrib. Comput.* 74, 10 (2014), 2967–2982.

[20] Kayo Fujiwara and Henri Casanova. 2007. Speed and Accuracy of Network Simulation in the SimGrid Framework. In *Proc. of the 2nd International Conference on Performance Evaluation Methodologies and Tools* (Nantes, France) *(ValueTools)*. ICST, Article 12, 10 pages.

[21] Cristian Galleguillos, Zeynep Kiziltan, Alessio Netti, and Ricardo Soto. 2020. AccaSim: A Customizable Workload Management Simulator for Job Dispatching Research in HPC Systems. *Cluster Computing* 23, 1 (2020), 107–122.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385

[23] Stephen Herbein, Dong H. Ahn, Don Lipari, Thomas R.W. Scogland, Marc Stearman, Mark Grondona, Jim Garlick, Becky Springmeyer, and Michela Taufer. 2016. Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters. In *Proc. of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*. ACM, 69–80.

[24] J. Hungershofer. 2004. On the combined scheduling of malleable and rigid jobs. In *Proc. of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. 206–213.

[25] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv:1602.07360

[26] iMatix. 2022. ZeroMQ—An open-source universal messaging library. https://zeromq.org/.

[27] INRIA, CNRS, ENS Rennes, and UH Mānoa. 2022. They use SimGrid. https://simgrid.org/usages.html.

[28] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 947–960. https://www.usenix.org/conference/atc19/presentation/jeon

[29] Zihan Jiang, Wanling Gao, Fei Tang, Xingwang Xiong, Lei Wang, Chuanxin Lan, Chunjie Luo, Hongxiao Li, and Jianfeng Zhan. 2021. HPC AI500: Representative, Repeatable and Simple HPC AI Benchmarking. arXiv:2102.12848

[30] Ana Jokanovic, Marco D'Amico, and Julita Corbalan. 2018. Evaluating SLURM Simulator with Real-Machine SLURM and Vice Versa. In *Proc. of the 9th IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 72–82.

[31] Youngjae Kim, Raghul Gunasekaran, Galen M. Shipman, David A. Dillow, Zhe Zhang, and Bradley W. Settlemyer. 2010. Workload characterization of a leadership class storage cluster. In *Proc. of the 5th Petascale Data Storage Workshop (PDSW)*. 1–5.

[32] Dalibor Klusáček, Šimon Tóth, and Gabriela Podolníková. 2016. Complex Job Scheduling Simulations with Alea 4. In *Proc. of the 9th EAI International Conference on Simulation Tools and Techniques (SIMUTOOLS)*. ICST, 124–129.

[33] Anthony Kougkas, Hariharan Devarajan, Xian-He Sun, and Jay Lofstead. 2018. Harmonia: An Interference-Aware Dynamic I/O Scheduler for Shared Non-volatile Burst Buffers. In *Proc. of the 20th IEEE International Conference on Cluster Computing (CLUSTER)*. 290–301.

[34] Alex Krizhevsky. 2012. Learning Multiple Layers of Features from Tiny Images. *University of Toronto* (2012).

[35] Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. arXiv:1404.5997

[36] Benjamin R. Landsteiner, Dave Henseler, Douglas Petesch, and Nicholas J. Wright. 2016. Architecture and Design of Cray DataWarp. https://cug.org/proceedings/cug2016_proceedings/includes/files/pap105s2-file1.pdf.

[37] Benjamin R. Landsteiner and David Paul. 2018. DataWarp Transparent Cache: Implementation , Challenges , and Early Experience. https://cug.org/proceedings/cug2018_proceedings/includes/files/pap119s2-file1.pdf.

[38] Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, and Carlos Maltzahn. 2012. On the role of burst buffers in leadership-class storage systems. In *Proc. of the IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*. 1–11.

[39] Gonzalo Martín, Maria-Cristina Marinescu, David E. Singh, and Jesús Carretero. 2013. FLEX-MPI: An MPI Extension for Supporting Dynamic Load Balancing on Heterogeneous Non-dedicated Systems. In *Proc. of the 19th International Conference on Parallel and Distributed Computing, Euro-Par 2013*, Felix Wolf, Bernd Mohr, and Dieter an Mey (Eds.). Springer, 138–149.

[40] Message Passing Interface Forum. 2021. *MPI: A Message-Passing Interface Standard Version 4.0.* https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

[41] Arash Partow. 2022. C++ Mathematical Expression Library. https://www.partow.net/programming/exprtk/index.html.

[42] Suraj Prabhakaran, Marcel Neumann, Sebastian Rinke, Felix Wolf, Abhishek Gupta, and Laxmikant V. Kale. 2015. A Batch System with Efficient Adaptive Scheduling for Malleable and Evolving Applications. In *Proc. of the 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 429–438.

[43] Martin Quinson, Cristian Rosa, and Christophe Thiery. 2011. Parallel Simulation of Peer-to-Peer Systems. In *Proc. of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 668–675.

[44] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. 2020. Designing Network Design Spaces. arXiv:2003.13678

[45] Gonzalo P. Rodrigo, Erik Elmroth, Per-Olov Östberg, and Lavanya Ramakrishnan. 2018. ScSF: A Scheduling Simulation Framework. In *Proc. of the 22nd Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, Dalibor Klusáček, Walfredo Cirne, and Narayan Desai (Eds.). Springer, 152–173.

[46] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4510-4520. (2018). arXiv:1801.04381

[47] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. arXiv:1802.05799

[48] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556

[49] Alan Jay Smith. 2007. Workloads (Creation and Use). *Commun. ACM* 50, 11 (nov 2007), 45–50.

[50] Massimo Benini Stephen Trofinoff. 2015. Using and Modifying the BSC Slurm Workload Simulator. https://slurm.schedmd.com/SLUG15/BSC_Slurm_Workload_Simulator_Enhancements.pdf.

[51] ThinkParQ and Fraunhofer ITWM. 2022. *BeeOND: BeeGFS On Demand.* https://doc.beegfs.io/latest/advanced_topics/beeond.html

[52] Marc-André Vef, Nafiseh Moti, Tim Süß, Tommaso Tocci, Ramon Nou, Alberto Miranda, Toni Cortes, and André Brinkmann. 2018. GekkoFS - A Temporary Distributed File System for HPC Applications. In *Proc. of the 20th IEEE International Conference on Cluster Computing (CLUSTER)*. 319–324.

[53] Pedro Velho, Lucas Schnorr, Henri Casanova, and Arnaud Legrand. 2013. On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations. *ACM Transactions on Modeling and Computer Simulation* 23, 4 (2013).

[54] Bing Xie, Jeffrey Chase, David Dillow, Oleg Drokin, Scott Klasky, Sarp Oral, and Norbert Podhorszki. 2012. Characterizing output bottlenecks in a supercomputer. In *Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC12)*. 1–11.

[55] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Proc. of the 9th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer, 44–60.