MalleTrain: Deep Neural Networks Training on Unfillable Supercomputer Nodes

Xiaolong Ma
University of Nevada, Reno
Reno, Nevada, USA
xiaolongm@nevada.unr.edu

Feng Yan
University of Houston
Houston, Texas, USA
fyan5@central.uh.edu

Lei Yang
University of Nevada, Reno
Reno, Nevada, USA
leiy@unr.edu

Ian Foster
Argonne National Laboratory
Lemont, Illinois, USA
University of Chicago
Chicago, Illinois, USA
foster@anl.gov

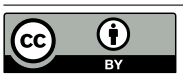
Michael E. Papka
Argonne National Laboratory
Lemont, Illinois, USA
University of Illinois Chicago
Chicago, Illinois, USA
papka@anl.gov

Zhengchun Liu
Argonne National Laboratory
Lemont, Illinois, USA
zhengchun.liu@anl.gov

Rajkumar Kettimuthu
Argonne National Laboratory
Lemont, Illinois, USA
University of Chicago
Chicago, Illinois, USA
kettimuthu@anl.gov

ABSTRACT

First-come first-serve scheduling can result in substantial (up to 10%) of transiently idle nodes on supercomputers. Recognizing that such unfilled nodes are well-suited for deep neural network (DNN) training, due to the flexible nature of DNN training tasks, Liu et al. proposed that the re-scaling DNN training tasks to fit gaps in schedules be formulated as a mixed-integer linear programming (MILP) problem, and demonstrated via simulation the potential benefits of the approach. Here, we introduce `MalleTrain`, a system that provides the first practical implementation of this approach and that furthermore generalizes it by allowing it to be used even for DNN training applications for which model information is unknown before runtime. Key to this latter innovation is the use of a lightweight online job profiling advisor (`JPA`) to collect critical scalability information for DNN jobs—information that it then employs to optimize resource allocations dynamically, in real time. We describe the `MalleTrain` architecture and present the results of a detailed experimental evaluation on a supercomputer GPU cluster and several representative DNN training workloads, including neural architecture search and hyperparameter optimization. Our results not only confirm the practical feasibility of leveraging idle supercomputer nodes for DNN training but improve significantly on prior results, improving training throughput by up to 22.3% without requiring users to provide job scalability information.



ICPE '24, May 7–11, 2024, London, United Kingdom

ACM ISBN 979-8-4007-0444-4/24/05.
https://doi.org/10.1145/3629526.3645035

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**; **Distributed algorithms**; • **Computer systems organization** → **Distributed architectures**.

KEYWORDS

Deep Neural Network, Distributed Deep Learning Training, Supercomputer, Scheduling, Resource Management

ACM Reference Format:
Xiaolong Ma, Feng Yan, Lei Yang, Ian Foster, Michael E. Papka, Zhengchun Liu, and Rajkumar Kettimuthu. 2024. MalleTrain: Deep Neural Networks Training on Unfillable Supercomputer Nodes. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3629526.3645035

1 INTRODUCTION

Batch-scheduled high-performance computing (HPC) systems typically maintain a queue of runnable jobs, with the order in which queued jobs are run being determined by resource scheduling policies established by administrators to meet higher-level goals. For example, the largest supercomputers often implement policies to encourage capability computing, wherein they prioritize large jobs that cannot run elsewhere. Other criteria, such as job wait time and recent usage by a user or group, may also be considered when determining job priorities. But regardless of policy goals, the fact that jobs are typically given exclusive access to a fixed number of nodes while running means that nodes will be idle whenever the number of free nodes is less than the number needed to run the next job (as identified by policy).

Backfilling [27], a method by which lower-priority, shorter, and/or smaller jobs are run on idle resources ahead of higher-priority jobs

as long as they do not delay the start time of the higher-priority jobs, can reduce, but not eliminate, inefficiencies, which can be substantial. For example, in 2012, a comprehensive analysis of a 12-month workload trace of the Kraken supercomputer showed an average utilization of 94% [37]; a four-year study of the Blue Waters system revealed that monthly utilization rarely exceeded 80%. Jones et al. [18]; and other studies have reported utilizations of around 90% [9, 25, 29]. These numbers can represent thousands of idle GPUs on large supercomputers.

One approach to enhancing utilization in such environments is to devise new approaches for structuring applications in `malleable forms` and for mapping these malleable applications to supercomputer resources. A malleable computation adapts its degree of parallelism at runtime in response to external requests [15], for example by using checkpointing for semi-automated stop/restart [34] or specialized languages and libraries [6, 11–13]. If well managed, malleable applications can improve system utilization and scheduling efficiency and reduce average response times, compared with unmalleable jobs. However, to realize these benefits, (a) malleable jobs need to be able to adapt dynamically to changing resource allocations and (b) job schedulers must be able to expand or shrink their resources to improve system utilization, throughput, and/or response times.

In practice, the rigid nature of both commonly used programming models like MPI and many current schedulers makes writing and running malleable applications a daunting task, which is why few malleable applications exist.

One intriguing source of malleable applications is deep neural network (DNN) training. DNNs are being employed widely in scientific computing [8, 10, 19, 21, 24, 26], and DNN training is becoming a major workload in today's supercomputers. Furthermore, deep learning frameworks such as AdaptDL [31], PyTorch TorchElastic [28], and Elastic Horovod [33] enable scaling up and down the number of workers dynamically during training at modest cost without requiring a restart. A DNN training job is divided into many smaller tasks (mini-steps) that can be fitted into node×time gaps in a supercomputer computing infrastructure. In other words, DNN training workloads can in principle be structured as malleable computations. However, practical realization of this malleability requires the ability to 1) determine, quickly and accurately, what mini-steps should be configured for different batch queue states, and 2) assign resources and computations to run those mini-steps.

Liu et al. recently showed how, given knowledge of scheduler state, the task of identifying mini-steps can be formulated as a deterministic mixed-integer linear programming–based resource allocation problem [25]. However, while they showed via simulation that this "FreeTrain" approach could construct effective schedules for real scheduler traces, they did not address the second task just listed, by providing a practical implementation of their proposed approach. This is a significant obstacle to the effective realization of malleable DNN training due to the need for several system components to coordinate and interact coherently: idle resource management, job progress monitoring, resource negotiation, and resource allocation. These components as well as their coordination are not readily available in today's job schedulers that were designed for unmalleable computing tasks.

A second deficiency of the FreeTrain approach is that it requires users to provide accurate scaling information, such as measured throughput when using different numbers of nodes for DNN training jobs. Providing this information is a substantial challenge because in many modern DNN training workflows, such as neural architecture search (NAS) [23, 32, 39] and hyperparameter tuning (HPO) [22], jobs are generated on the fly based on results produced in previous iterations by methods such as reinforcement learning [39] and Bayesian optimization [14]. Thus, even experienced DNN experts do not know all the model details beforehand, let alone their scalability characteristics.

In the work reported here, we propose and demonstrate solutions to the two obstacles to the practical realization of malleable DNN training just noted. First, we present a malleable DNN training system architecture, `MalleTrain`, which achieves the efficient coordination of the required idle resource management, job progress monitoring, resource negotiation, and resource allocation functions. For instance, in order to make malleable scheduling decisions, the Resource Allocator must first get information about unfillable nodes from the batch scheduler (e.g., PBS [16] or Slurm [36]), profiling information from a profiler, and current running and waiting DNN jobs from the job monitor; then, it needs to control a DNN scaling framework (e.g., Elastic Horovod) to execute the scheduling decisions. Throughout this process, it must also avoid negative impacts on jobs submitted to the main batch scheduler.

Second, we address the challenge of obtaining accurate scaling information by introducing a lightweight job profiling advisor (`JPA`) to obtain automatically the information required for making resource management decisions. `JPA` runs experiments whenever a DNN training task starts, according to a schedule that minimizes associated costs by taking advantage of the fact that removing a node is faster than adding a node in distributed DNN training. By thus obtaining accurate job information at modest cost, `JPA` permits the MILP to make more accurate decisions, with significant benefits in practice. We conducted extensive simulation evaluations using workloads from production supercomputer clusters, alongside experiments on a smaller cluster with synthetic logs derived from real Summit cluster logs. Our findings indicate that the more accurate information provided by `JPA` allows `MalleTrain` to achieve performance improvements of up to 22.3% relative to FreeTrain. In addition, it permits the scheduling of malleable DNN training applications, such as NAS and HPO, for which no performance information may be available.

This paper thus makes three important contributions. First, we propose a system architecture for running malleable DNNs on supercomputers, and implement `MalleTrain` according to this architecture. Second, we propose a lightweight online profiler that employs an inverse-order profiling method to obtain accurate scalability information for dynamic DNN jobs. Third, we present results from both simulations with supercomputer traces and real-world executions on a cluster using synthetic traces that demonstrate the efficiency of these methods in harnessing previously idle nodes for DNN training—and thus the feasibility of using what may often be 10% or more of previously unfillable supercomputer nodes for large-scale DNN training.

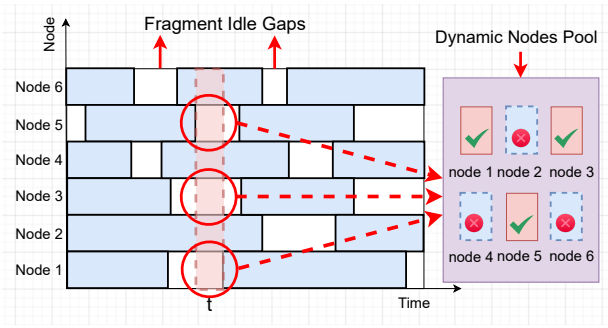


Figure 1: Illustration of dynamic fragment resources on a portion of a cluster. At time t, there are three idle nodes in the `MalleTrain` resource pool.

2 BACKGROUND

We present background information on the methods used by cloud providers to support malleability, fragmented resources in HPC, FreeTrain, and HPC network topologies.

2.1 Cloud-Preemptable Instances

Cloud providers such as AWS [1], Google [2], and Azure [3] make preemptable compute capacity available at a reduced cost via mechanisms such as AWS Spot Instances. For AWS, Spot Instances enable strategic utilization of surplus capacity; for users, they provide an opportunity to reduce their cloud expenses. To make use of such resources, however, users must be flexibile in their application runtime and tolerance for interruptions.

Spot Instances are particularly well suited for certain noncritical tasks such as data analysis, batch processing, and background operations. As noted, their costs are typically lower than for regular instances; on the other hand, they do not provide a time guarantee, introducing the possibility of unexpected interruption due to the cloud provider reclaiming running instances. To mitigate the potential impact of such interruptions, cloud providers often grant a brief time window and prior notification to clients. This advance notice enables clients to reconfigure their workload distribution, effectively rebalancing the workload across available resources. By reallocating tasks and resources in response to an impending reclamation, clients can minimize disruptions and maintain a good level of user experience. Spot Instance resources in cloud environments resemble the preemptible HPC nodes addressed by `MalleTrain`.

2.2 Fragment Resources on HPC

As explored in recent research [9, 25, 29], leadership supercomputer clusters such as Mira, Theta, and Summit exhibit utilization rates of around 90%. Considering the substantial scale of these leadership supercomputer clusters, the unutilized resources become a significant concern. To put this in perspective, 10% idle capacity corresponds to 460 nodes on the 4608-node Summit and more than 1000 nodes on the 10,624-node Aurora.

Resource allocation within supercomputer clusters is typically managed by main schedulers such as Slurm [36] or PBS [16]. These schedulers administer multiple queues to prioritize resource assignments for user requests. As depicted in Figure 1, inevitable fragmentary resources emerge. These fragments may not always be backfilled, and (a portion of them) may remain unassigned. However, these seemingly negligible fragments are well suited for scalable and/or fault-tolerant workloads. The nature of these unassigned fragment resources resembles that of Spot VMs, as discussed in §2.1. In subsequent sections we will refer to these fragment resources within supercomputer clusters as *preemptible nodes*. Their allocation timing lacks guarantees, rendering them unsuitable for typical fixed-size supercomputer workloads. Nonetheless, the paradigm of malleable applications, exemplified by DNN training, aligns seamlessly with this computational context. This suitability is underscored by several key factors: (1) DNN training demands substantial time and computational resources; (2) the distributed data-parallel training paradigm is inherently scalable; (3) leading DNN training frameworks, such as Horovod Elastic [33] and TorchElastic [28], adeptly support elastic training; and (4) DNN training often involves exhaustive searches for optimal neural network architectures and hyperparameters, consuming extensive computational resources.

The objective of `MalleTrain` is to empower users to effectively leverage the unfilled fragments in supercomputers. Some supercomputers have a preemptable queue (the jobs submitted to this queue may be preempted anytime) explicitly to encourage the use of the unfilled nodes. A preemptable queue can be designated for `MalleTrain` to which the users will submit adaptable DNN training jobs. `MalleTrain` will optimally manage the allocation of unfilled nodes by dynamically expanding and shrinking these adaptable DNN jobs. To incentivize the adoption of this preemptable queue, benefits such as reduced charges, in terms of either monetary cost or node-time consumption, can be extended to users.

Table 1: Queue types and their characteristics. *Queue* is the queue type name on the Polaris cluster, the *Min* and *Max* columns give minimum/maximum number of nodes, and time, allowed per job request, and *Priority* is the priority for jobs in the queue.

	Nodes		Time		
Queue	Min	Max	Min	Max	Priority
debug	1	2	5 min	1 hr	debug
debug-scaling	1	10	5 min	1 hr	debug
demand	1	56	5 min	1 hr	High
prod	10	496	5 min	24 hr	High
preemptable	1	10	5 min	72 hr	Low

Table 1 displays the different queue types in the Argonne Leadership Computing Facility (ALCF) Polaris cluster [4]. The low job priority means that nodes allocated for the job in the preemptable queue could be reclaimed.

2.3 FreeTrain

As noted earlier, FreeTrain [25] introduces an approach to dynamically allocating idle resources in which nodes and running job information are taken as inputs and user-defined metrics such as throughput or scalability are adopted as optimization objectives. By formulating the problem using MILP, FreeTrain is able to compute

an optimal allocation of idle resources to DNN training jobs, subject to constraints such as allowed job size, feasible resource allocation, job scale information, and job migration overhead.

However, several practical challenges must be overcome before this approach can be realized into a production environment:

(1) **Expecting users to provide specific runtime job details can be a significant burden to users.** The MILP algorithm requires users to supply precise job-specific information, such as model training throughput and scalability, since these details serve as essential inputs for the optimization process. This requirement will significantly increase the burden on users.

(2) **Job runtimes often correlate closely with specific hardware capability and configurations.** Thus, to attain accurate job runtime information, users would have to prerun their jobs under nearly identical system settings and hardware configurations. However, this approach would be prohibitively time-consuming and resource-consuming for most supercomputer users.

(3) **In some cases, heuristic algorithms rely on current models to predict future executions, making it impractical to preprofile all potential models.** The majority of HPO/NAS algorithms are heuristic [14, 23, 32, 39], which implies that the models to be evaluated are not predetermined until the current models have completed their execution. Thus, users will not be able to provide accurate job runtime information, a situation that will lead to an invalid resource allocation plan and will largely downgrade the performance of the system.

To overcome these challenges, an intelligent online profiling mechanism is needed. Such a mechanism should accurately collect job runtime information while minimizing disruptions to the regular execution of jobs.

`MalleTrain` also employs MILP to do the allocation optimization but emphasizes practical deployment aspects in supercomputer clusters. JPA can be integrated seamlessly into the workflow, orchestrating automatic profiling and obviating the need for manual input. As a result, the profiling procedure becomes an inherent facet of the process, efficiently alleviating the user from the need to provide such details beforehand. This dynamic profiling process operates in real time, eliminating the need to halt any ongoing jobs. While the profiling phase may occasionally lead to suboptimal cluster performance, we mitigate potential overhead through the implementation of a carefully designed online profiling mechanism. Thus our design is able to obtain accurate profiling information without excessive operational costs.

2.4 Topology

The network topology in a supercomputer cluster plays an important role in facilitating efficient communication, seamless data transfer, and effective management of network resources. Today, the dragonfly [20] and fat-tree [7] topologies are widely utilized in supercomputer clusters due to their ability to deliver high bandwidth and low latency. These features make them adept at meeting the demanding requirements of modern high-performance computing environments. The Polaris cluster and upcoming Aurora cluster in the ALCF both use the dragonfly network topology, and the Summit cluster uses fat-tree. A major concern for fragmented idle resources in a supercomputer is that such resources will often

Figure 2: Example of fragment resources distribution on Polaris (27th in the TOP500 supercomputer list on Nov. 2023). Red stars mark fragmented idle resources scattered on the cluster. Note: For clarity in presentation, the figure depicts a majority of the cluster rather than its entirety.

be scattered and distant from each other, as shown in Figure 2. Each color represents a job; the nodes with same color were allocated to the same job. To fully utilize the inter connection bandwidth and reduce the latency, schedulers tend to assign the nodes into the same group or make them close to each other. For fragmented resources, however, usually the nodes are scattered into different topology groups. This scattering will have two major impacts. First, long distance usually means more hops are needed, which means the connections could have a higher fluctuation and cause a downgrade in the DNN training performance. Second, long distance could increase the end-to-end latency and cause more network resource contentions. We perform extensive evaluation and show that the topology is not a critical bottleneck for the design of `MalleTrain`.

3 SYSTEM DESIGN AND REALIZATION

`MalleTrain` manages the residual resources of a supercomputer cluster, in other words, those that at any particular moment have not been allocated directly by the main scheduler. Two major challenges for `MalleTrain` arise in utilizing such residual resources: (1) their availability varies dynamically, and (2) they are preemptible. The `MalleTrain` design enables these resources to be utilized fully for parallel DNN training. `MalleTrain` seamlessly integrates with mainstream schedulers such as Slurm or PBS on supercomputer clusters. It operates without impacting the main scheduler, exclusively controlling the non-trivial, dynamic, residual resources that the main scheduler cannot utilize.

3.1 System Architecture Overview

Figure 3 shows the `MalleTrain` architecture and its five primary components, which we describe in the following:

Scavenger detects and collects idle nodes from the main job scheduler for `MalleTrain`. Two primary approaches could be employed: an event-driven mechanism, whereby the main scheduler alerts `MalleTrain` to idle nodes, or a proactive strategy, in which Scavenger periodically polls to find available unused (but preemptible when the main scheduler needs them) resources. The latter approach, preferred for its autonomy, requires no additional action

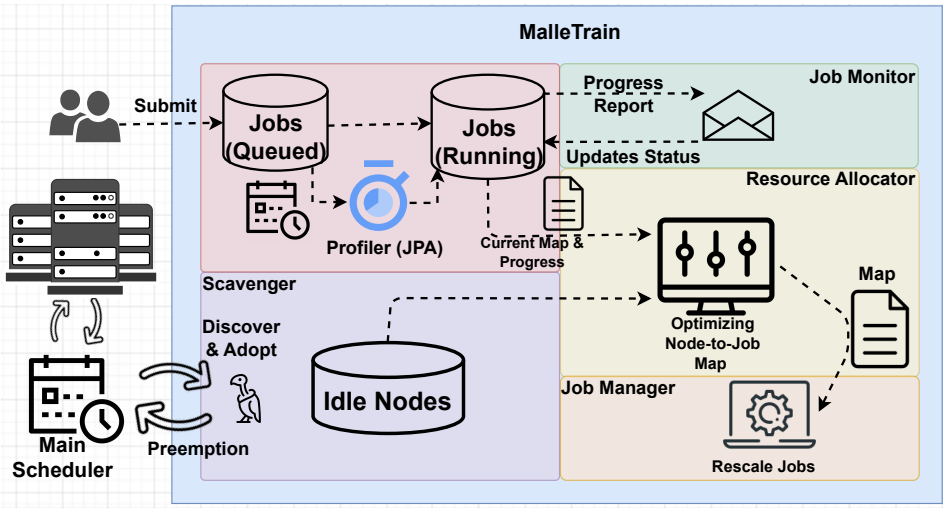


Figure 3: Schematic of the `MalleTrain` architecture. Scavenger adopts idle nodes, Resource Allocator determines a map of nodes to jobs, Job Manager rescales jobs according to the map, Job Monitor tracks job progress, and Job Profiling Advisor manages the online profiling process.

from the main scheduler, ensuring seamless and efficient use of idle nodes by `MalleTrain`.

Resource Allocator maps nodes to DNN jobs in such a way as to optimize a given metric such as throughput or scaling efficiency. The allocation task can be formulated as a mathematical programming problem. In this paper we adopt the formulation of Liu et al. [25] for resource allocation. The Resource Allocator is event-driven, with four types of events being considered: new nodes joining `MalleTrain`, nodes being recalled by the batch scheduler (i.e., the corresponding jobs are preempted), arrival of new `MalleTrain` jobs, and `MalleTrain` jobs completing.

Job Manager manages all jobs and implements the jobs-to-nodes mapping made by the Resource Allocator.

Job Monitor tracks job progress by consuming (current global batch size, timestamp) records generated by DNN training jobs via one line of `MalleTrain`-supplied code added to the training loop. The Monitor module then computes the current throughput as well as the cost incurred for each rescale operation and updates that information in a job records table to be used by the Resource Allocator.

Job Profiling Advisor manages the online profiling process, as described in §3.3. The JPA is an independent component that starts work before the job entering the Resource Allocator.

When nodes cannot be backfilled by the main scheduler, they are redirected to the Scavenger for utilization. Jobs submitted by users to `MalleTrain` await the availability of nodes. As nodes become available, the jobs at the front of the queue commence execution. The running jobs transmit progress updates to the Job Monitor via a socket client. The system's architecture ensures continuous reporting of both cluster node statuses and job execution information to the Resource Allocator. The Allocator then employs MILP based on the current job distribution and number of nodes in the Scavenger. The MILP algorithm devises a strategic plan, which is represented by a map and subsequently conveyed to the Job Manager. The Job Manager then implements this plan to adjust resources accordingly. The events described in §3.2 will trigger the Resource Allocator to run MILP and generate a new adjustment plan.

Users are provided with the option to explicitly indicate whether their job requires profiling. If so, the JPA consults with the Resource Allocator to assess the availability of necessary node resources for profiling. Should resources be insufficient, the jobs are returned to the queue. Conversely, if adequate resources are available for profiling, the job proceeds through the profiling process. This process uniquely involves an inverse order of node numbers; further details are given in §3.3. When the profiling process is done, the profiled job information will be an input to the MILP to find the optimal allocation.

3.2 Event-Driven Resource Adjustment

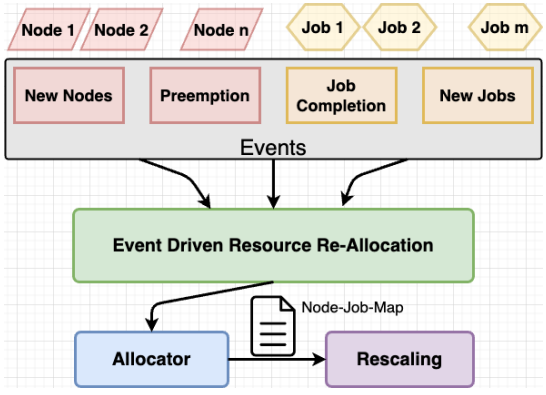


Figure 4: Event-driven resource allocation process.

Our event-driven resource management architecture is shown in Figure 4. There are four types of events:

New Nodes indicates that one or more nodes have become available to `MalleTrain`.

Preemption can be initiated at any time by the main scheduler without any prior notification. The jobs being run by `MalleTrain` on the preempted nodes are terminated and the nodes returned to the main batch scheduler.

Job Completion. `MalleTrain` picks a maximum of the top (first come, first serve, FCFS) jobs from its queue to prevent excessive hunger of low-priority jobs (e.g., low-throughput jobs when sample processed per second is the target to optimize). All the selected jobs are launched by `MalleTrain` via spawning a process using a `subprocess` module of Python in a nonblocking fashion. The exit/completion of a job is thus notified from the Job Monitor module of `MalleTrain`.

A **New Jobs** event can trigger resource allocation only when the number of currently running jobs, $Njrun$, is less than the jobs number threshold allowed in `MalleTrain`, $Pjmax$. When more than one job is submitted as a batch (e.g., grid search of a hyperparameter search), $Pjmax - Njrun$ jobs will be added to the running list as a batch to reduce the rescaling cost. When the number of arrving jobs $Njarrive$ is larger than $Pjmax - Njrun$, the $Njarrive - (Pjmax - Njrun)$ jobs will be put into the FCFS queue for future execution.

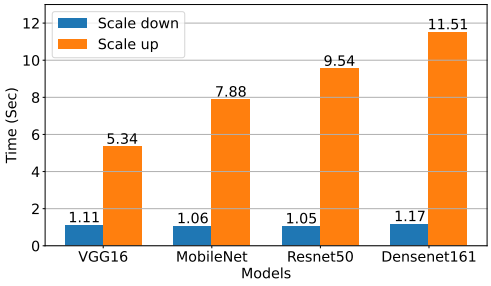
Table 2: Example jobs-to-nodes map, as determined by MILP. Each row corresponds to a job, with scale given by the sum of the cells in the row; each column corresponds to a node, with at most one cell in the column with value 1 indicating the job to which the node is allocated.

	N_1	N_2	N_3	N_4	N_5	N_6	N_7	...	...	N_n
J_1	0	0	1	0	0	0	1	0	0	0
J_2	0	0	0	0	0	0	0	0	1	1
...	1	0	0	1	1	0	0	0	0	0
J_4	0	1	0	0	0	1	0	1	0	0

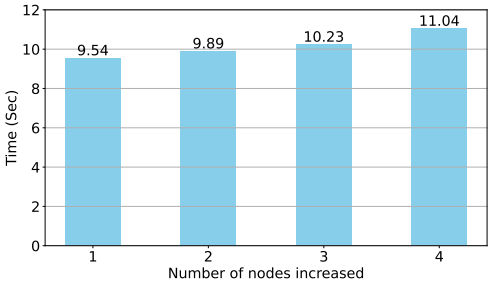
The node-job map shows the allocation plan, and Table 2 demonstrates an example map of the allocation plan. The MILP optimizer takes the input and gives a new node-job map to the Allocator to do the reallocation. We give more details in §3.3.

3.3 Job Profiling Advisor

In contrast to traditional profiling methods that necessitate dedicated resources, our online profiling process is integrated into the training process. This approach ensures the uninterrupted operation of worker processes during profiling. The strategic design of node adjustment sequences, as depicted in Figure 6, to avoid scale-up operations, effectively minimizes additional overhead. Each job is equipped with a lightweight reporter (socket client), responsible for reporting job progress to the Job Monitor (socket server). This approach facilitates the automatic aggregation by the Job Monitor of the training process information that is then used for optimization purposes. Consequently, the JPA is enabled to make precise and timely adjustments, thereby maximizing resource utilization.



(a) 1-node scale-up and scale-down costs.



(b) Scale-up times vs. number of nodes: ResNet-50.

Figure 5: Rescaling overhead costs on Polaris A100 GPU nodes: (a) Time to scale up and down a single node, for different models; (b) Time to scale up different numbers of nodes, for ResNet-50 model.

We noted in §2.3 the necessity for online profiling in order to permit accurate MILP solutions and to handle tasks for which profile information is not available before their execution. Here we shift focus to an in-depth examination of the design elements of JPA. In our proposed design the profiling function runs concurrently with jobs. Thus we want it to be:

Prompt, meaning that it processes profiling events rapidly so as to ensure efficient utilization of profiling information, and furthermore completes rapidly so as to minimize overhead and limit disruption to other tasks;

Fair, meaning that its design incorporates principles of fairness, and that in instances where job interruption is unavoidable, a Least Recently Used (LRU) strategy is employed to ensure equitable distribution of interruptions; and

Efficient, meaning that it prioritizes minimal disruption to other tasks, adhering to two key principles: (1) avoiding the interruption of multiple jobs simultaneously and (2) preventing the complete cessation of any single job.

Accurate MILP requires that we know, or can rapidly determine, the time that will be required to run any training mini-task on any possible number of nodes. While obtaining this information may sound intractable, in practice the regular nature of DNN computations makes it feasible to obtain good estimates. As in FreeTrain, we assume a fixed per-node minibatch size (when training, we employ a learning rate scheduler to adjust learning rate according to the

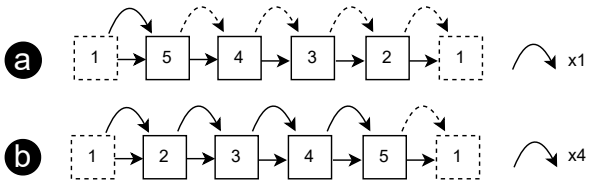


Figure 6: Inverse-order rescaling sequence. The solid curve represents scale-up and the dashed curve scale-down. JPA aims to minimize the number of scale-up operations in order to reduce overhead.

global batch size [17, 38]). We then need simply to measure the time per epoch for that minibatch size on different numbers of nodes, from a specified minimum to a specified maximum.

A useful optimization when performing those measurements derives from the observation that, as shown in Figure 5a, the cost of scaling up is consistently multiple times greater than that of scaling down. Furthermore, Figure 5b illustrates that the overhead incurred during scale-up remains relatively constant regardless of the number of nodes involved; even as the number of nodes increases, the increase in scale-up time is marginal. Consequently, in our profiling of the rescaling process, we should minimize the need for scaling up and prioritize scaling down wherever feasible. As an example, consider the two situations illustrated in Figure 6. If the initial number of nodes is 1 and the objective is to profile nodes 2, 3, 4, and 5, we may either: (a) scale up directly to 5 nodes and then scale down to 1, thereby gathering scalability data for all nodes using a single scale-up operation, or (b) incrementally scale up from 1 to 5, which requires four separate scale-up operations. The first approach is significantly more efficient than the first, since it requires only one scale-up operation.

The JPA architecture (Figure 7) resembles that of `MalleTrain` but with several distinctions: (1) JPA exclusively processes new job events, since only these require profiling; in contrast, the trainer instance accepts multiple events, as described in §3.2. (2) The node adjustment in JPA is decided by our profiling algorithm instead of by the MILP program. Users retain the discretion to decide whether their jobs undergo profiling. Upon receiving a profiling request from a user, a profiling event is triggered, which initiates a process whereby the Resource Allocator assesses the availability of sufficient resources for profiling. If resources are deemed adequate, a profiling job is started, temporarily preempting nodes from other jobs. Upon completion of profiling, the MILP process is engaged to make adjustments based on the newly collected information. The gathered scale information is then reported and recorded by the job manager, contributing to future optimization efforts.

3.4 Cluster Configuration

MILP is an NP-hard problem and the cost of the MILP computation required to determine a mapping of jobs to idle nodes scales rapidly with the number of runnable jobs and available nodes. Thus, it can be preferable to partition a supercomputer into disjoint subsets and run multiple trainers in parallel, one per subset. This approach restricts the maximum number of nodes to which any one job can scale, but has the advantages of reducing delays due to training and of permitting different trainers to optimize for different metrics appropriate for different task types, such as computer vision models and language models.

With multiple trainers, the question arises of whether it is advisable from a performance perspective to run more than one on a single node. Our preliminary investigation into the effects of running multiple MILP processes concurrently on the same node revealed that the processing time begins to increase only when the number of concurrent trainers exceeds the number of cores, as illustrated in Figure 8. This suggests that deploying multiple trainers and running the associated MILP processes concurrently on a single head node can diminish overheads without adversely affecting the performance of standard jobs.

4 EVALUATION AND DISCUSSION

We conducted an extensive experimental evaluation to validate the effectiveness and robustness of our framework with real logs of supercomputer clusters. We also validated `MalleTrain` on a small cluster in a real production environment.

4.1 Experiment Setup

We examined trace logs from two supercomputers listed in the TOP500 as of November 2023: Summit, ranked 7th, and Polaris, ranked 27th [5]. The Summit log spans 14 days from February 10 to February 24, 2021, while the Polaris log covers a 7-month duration, from January 1 to July 28, 2023.

Figure 9 depicts event traces from the Summit and Polaris supercomputers. We see that Polaris has more shorter gaps than Summit, with indeed over 50% of its event gaps being shorter than 10 seconds. A key factor contributing to this difference is Summit's policy favoring large jobs. Such jobs generally have longer durations, leading to fewer but more extended resource occupations. Conversely, without a similar policy favoring large jobs, Polaris experiences more frequent, shorter gaps between events due to the prevalence of smaller jobs. However, because of the unavailability of idle node data for the Polaris cluster, we focus on Summit trace data in our log replay simulation evaluation. Figure 10, which shows idle nodes on Summit over a two-week period, shows that the number of idle nodes varies significantly over time.

While plugging `MalleTrain` into the batch scheduler of a real supercomputer would permit accurate evaluation in a real system, we would lose the ability to reproduce the same trace with different strategies, including the baseline allocation policy, for comparative research. Therefore, we instead generate representative traces and replay them on the real system for our experimental evaluation. In contrast to the simulation-based evaluation, experiments here do not rely on any performance modeling: they run the DNN training task on real supercomputer nodes.

A challenge for `MalleTrain` is to optimally utilize fragmented node×time resources to meet a user-specified metric (e.g., throughput in terms of samples processed per second, resource utilization/scaling efficiency). We synthesize traces that are independent and identically distributed with real traces from supercomputers. Figure 11 compares node idle gap lengths from real Summit scheduler logs vs. our synthetic traces. We see that the distribution of

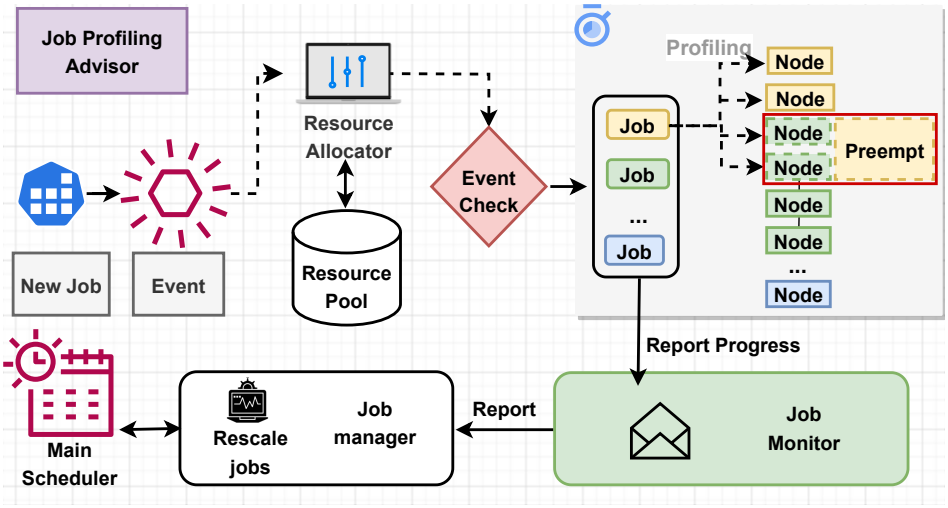


Figure 7: Online profiling process.

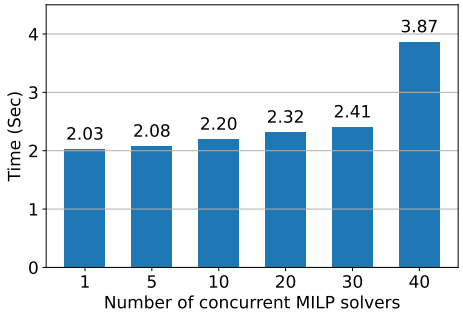


Figure 8: Average time taken for an example MILP computation as the numbers of concurrent MILP computations performed on a single 32-core head node scales from 1 to 40.

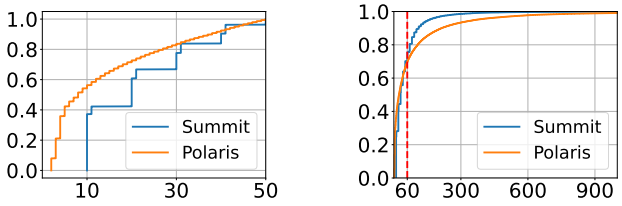


Figure 9: Cumulative histograms of idle gap counts on Summit and Polaris, for short gaps (0–50 secs: left) and longer gaps (0–3600 secs: right). Polaris has more shorter gaps (≤60 secs) while Summit has more gaps in the range from 60 to 600 secs.

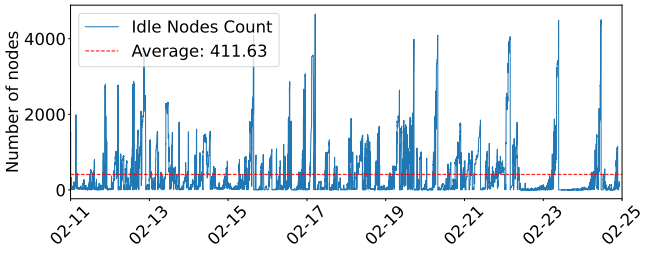


Figure 10: Idle nodes on Summit over two-week period.

synthetic traces is close to those of the real logs, confirming the representativeness of our synthetic traces.

4.1.1 Workload. NASBench101 [35] is a neural architecture search (NAS) benchmark dataset created to permit systematic, reproducible, and accessible evaluation of NAS algorithms. It was introduced to address the challenges associated with the high computational cost of evaluating NAS algorithms, which traditionally require training thousands of neural network architectures from scratch to find the most efficient one for a given task. We conducted our experiment within the search space of NASBench101, which comprises 423,624 computationally unique neural architectures. The image size for our training is 224×224×3. We use randomly generated tensors instead of the real dataset to remove the potential I/O impact on our experiments. We note that our focus here is not on the accuracy of the models but rather on assessing throughput and scalability. Varieties of deep learning models that do the HPO tasks were also evaluated in the same context as the NAS workload; the models were randomly selected from models listed in Figure 14.

4.1.2 Testbed. We conducted experiments on a 32-node cluster in which each node is equipped with four A100 GPUs. The GPUs are interconnected via NVLink within each node, and nodes are connected via InfiniBand. The synthesized traces, as depicted in Figure 11, were instrumental in simulating the preemptive actions undertaken by the main scheduler.

4.2 Performance Evaluation

We conducted experiments to benchmark our system against the FreeTrain framework for preemptible resource allocation on HPC

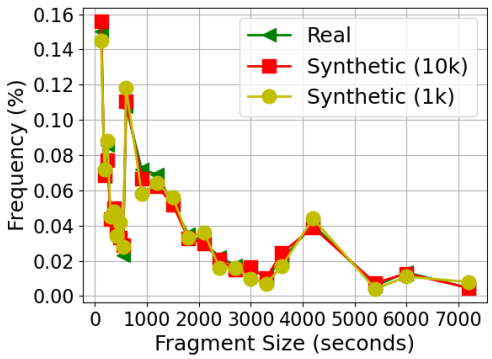


Figure 11: Comparison histogram of fragment length between real logs and synthetic. Synthetic (10k) shows the statistics for 10k fragments, and Synthetic (1k) shows the statistics for 1k fragments. The synthetic traces keep the same distribution as that of the real log.

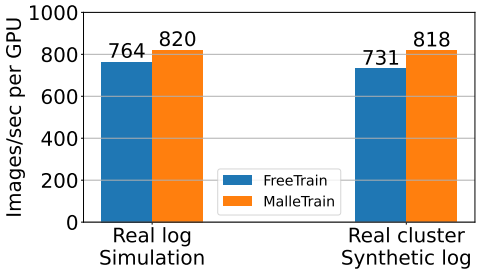
clusters. Our evaluation comprised NAS and HPO training workloads. Notably, the NAS workload exhibited more variability in training speed and scalability compared with HPO tasks.

Our primary metric for comparison was the overall training throughput of the system. We ran both frameworks under identical workloads to ensure a fair comparison. For the NAS model sampling process, we randomly selected models. To maintain consistency, we set the same seed value for both frameworks, ensuring that the sequence of model training remained identical across the experiments. We conducted the simulation with the two-week log and conducted the experiments for 12 hours with the synthetic trace. The average throughput is shown in Figure 12 with the NAS workload and HPO workload. We see that `MalleTrain` outperforms FreeTrain in various settings.

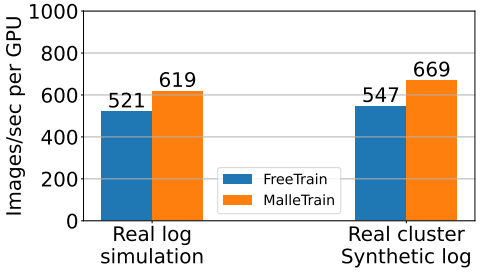
4.3 Topology Impact Analysis

The dynamic and randomly scattered nature of fragmented resources across the cluster raises concerns about potential declines in the overall performance of training jobs. To address these concerns, we conducted experiments on the Polaris cluster with the dragonfly network topology. Our study involved comparing the performance of nodes confined within a single dragonfly group versus those distributed across multiple dragonfly groups. Figure 13 shows that the physical distribution of nodes, whether scattered or closely situated, has minimal impact on NAS/HPO DNN training speed. Figure 14 indicates robust scalability of models even at the 32-node level, each node equipped with 4 NVIDIA A100 GPUs, encompassing a total of 128 A100 GPUs.

The underlying reasons for these observations are multifaceted. First, leadership-class supercomputer clusters are typically outfitted with high-performance network devices. For instance, Polaris is equipped with the HPE Slingshot 11 interconnect, offering up to 200 Gb/s point-to-point bandwidth. Second, the networking infrastructure in these clusters is often highly overprovisioned, mitigating network contention among applications running on different nodes.

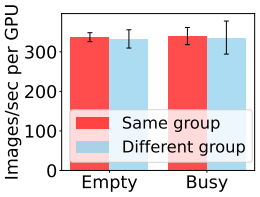


(a) Neural architecture search

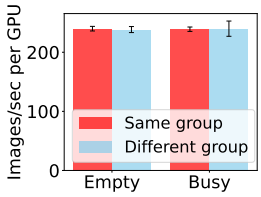


(b) Hyperparameter optimization

Figure 12: FreeTrain vs. `MalleTrain` performance for the NAS and HPO applications, as measured both with real logs on a simulator and synthetic logs on a real cluster.



(a) ResNet-50



(b) VGG-19

Figure 13: We analyzed training performance for sample `MalleTrain` jobs under four different scenarios: *Same Group, Empty* (where all nodes are located within the same Dragonfly group and the cabinet is empty), *Same Group, Busy* (where all nodes are within the same Dragonfly group but are collocated with other jobs), *Different Group, Empty* (where nodes are distributed across two Dragonfly groups with the two cabinets empty), and *Different Group, Busy* (where nodes are distributed across two Dragonfly groups and collocated with other jobs). The results demonstrate consistent training speeds for both models across all scenarios. The error bars for the *Different Group, Busy* scenario reveal higher variances in training speed, indicating fluctuations occur primarily in this scenario. However, the average training speed remains consistent despite these fluctuations.

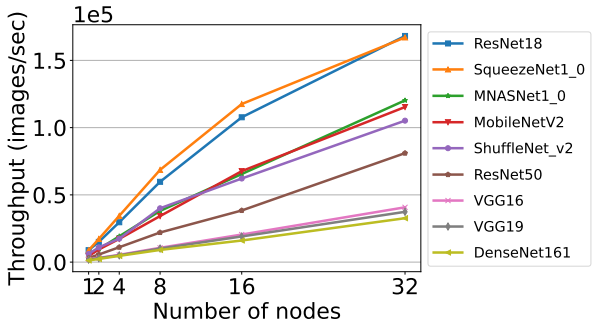


Figure 14: Trend analysis of model scalability on 32 Polaris nodes, each with 4 A100 GPUs.

Third, modern distributed deep learning frameworks, such as PyTorch [28] and Horovod [33], effectively overlap computing and communication tasks. This overlapping functionality reduces the network's impact on training speed, thereby diminishing the sensitivity to network conditions.

5 RELATED WORK

We have already referred to the pioneering work of Liu et al. on FreeTrain [25], while noting also that certain assumptions and strict requirements make it fall short in the real production environment. FreeTrain heavily relies on users to provide accurate runtime information from training jobs, which increases the burden to the users, making it impractical for use. Indeed, in some widely used heuristic NAS/HPO algorithms, FreeTrain has to guess a configuration or provide information solely based on user experience; the inaccurate or out-of-date information might largely downgrade the overall performance of the MILP optimization algorithm. In contrast, `MalleTrain` integrates automatic profiling components into the process and doing the profiling automatically.

Pollux [30] is a resource-adaptive DNN training and scheduling framework designed to efficiently rearrange distributed deep learning processes, particularly in dynamic-resource environments such as shared clusters and cloud infrastructures. This framework employs Kubernetes for efficient scheduling, rescaling, and reconfiguring of job batch sizes and learning rates, thus maximizing training performance and optimizing resource utilization. Pollux operates on a fixed-size cluster, however, whereas `MalleTrain` can handle dynamically varying cluster sizes.

6 CONCLUSION

We have introduced `MalleTrain`, a system that we demonstrate can employ idle fragmented nodes on batch-scheduled HPC systems for large-scale DNN training. `MalleTrain` defines a workable architecture for efficient use of such idle nodes, and via its job profiling advisor, which efficiently gathers accurate job execution data at runtime with minimal interference to ongoing tasks, enables idle nodes to be employed efficiently even for dynamic workloads such as neural architecture search and hyperparameter optimization. Detailed performance studies involving both simulations and experiments validate the effectiveness of the approach and show that `MalleTrain` achieves >20% more training throughput than was reported, on the basis of simulation studies alone, for a precursor system. `MalleTrain` thus opens up the feasibility of both improving the utilization of large HPC systems and increasing the resources delivered to DNN applications. Moreover, the methodologies developed in this study have potential applications beyond their current scope. They could be adapted, for example, to infrastructure management tasks, such as scheduling in Kubernetes clusters and other cloud computing platforms.

ACKNOWLEDGMENT

We are grateful to the reviewers for their valuable feedback and comments and Gail Pieper for helping us edit our paper. This work was sponsored in part by NSF grant CAREER-2048044 and by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

REFERENCES

[1] 2023. https://aws.amazon.com/ec2/spot/. Accessed: 2023-10-23.
[2] 2023. https://cloud.google.com/spot-vms. Accessed: 2023-10-23.
[3] 2023. https://azure.microsoft.com/en-us/products/virtual-machines/spot/. Accessed: 2023-10-23.
[4] 2023. https://www.alcf.anl.gov/polaris. Accessed: 2023-10-23.
[5] 2023. https://www.top500.org/lists/top500/2023/11/. Accessed: 2023-11-15.
[6] Bilge Acun, Abhishek Gupta, Nikhil Jain, Akhil Langer, Harshitha Menon, Eric Mikida, Xiang Ni, Michael Robson, Yanhua Sun, Ehsan Totoni, et al. 2014. Parallel programming with migratable objects: Charm++ in practice. In *SC'14: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 647–658.
[7] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review* 38, 4 (2008), 63–74.
[8] Ahsan Ali, Hemant Sharma, Rajkumar Kettimuthu, Peter Kenesei, Dennis Trujillo, Antonino Miceli, Ian Foster, Ryan Coffee, Jana Thayer, and Zhengchun Liu. 2022. fairDMS: Rapid model training by data and model reuse. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 394–405.
[9] William Allcock, Paul Rich, Yuping Fan, and Zhiling Lan. 2017. Experience and practice of batch scheduling on Leadership Supercomputers at Argonne. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1–24.
[10] Selin Aslan, Zhengchun Liu, Viktor Nikitin, Tekin Bicer, Sven Leyffer, and Doga Gursoy. 2020. Distributed optimization with tunable learned priors for robust ptycho-tomography. *arXiv preprint arXiv:2009.09498* (2020).
[11] Sebastian Buchwald, Manuel Mohr, and Andreas Zwinkau. 2015. Malleable Invasive Applications. In *Software Engineering (Workshops)*. 123–126.
[12] Ewa Deelman, Karan Vahi, Mats Rynge, Rajiv Mayani, Rafael Ferreira da Silva, George Papadimitriou, and Miron Livny. 2019. The evolution of the Pegasus workflow management software. *Computing in Science & Engineering* 21, 4 (2019), 22–36.
[13] Travis Desell, Kaoutar El Maghraoui, and Carlos A Varela. 2007. Malleable applications for scalable high performance computing. *Cluster Computing* 10, 3 (2007), 323–337.
[14] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*. PMLR, 1437–1446.
[15] Dror G Feitelson and Larry Rudolph. 1996. Toward convergence in job schedulers for parallel supercomputers. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1–26.
[16] Hanhua Feng, Vishal Misra, and Dan Rubenstein. 2007. PBS: a unified priority-based scheduler. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and Modeling of Computer Systems*. 203–214.
[17] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[18] Matthew D Jones, Joseph P White, Martins Innus, Robert L DeLeon, Nikolay Simakov, Jeffrey T Palmer, Steven M Gallo, Thomas R Furlani, Michael Showerman, Robert Brunner, et al. 2017. Workload analysis of Blue Waters. *arXiv preprint arXiv:1703.00924* (2017).

[19] Julian Kates-Harbeck, Alexey Svyatkovskiy, and William Tang. 2019. Predicting disruptive instabilities in controlled fusion plasmas through deep learning. *Nature* 568, 7753 (2019), 526–531.

[20] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. 2008. Technology-driven, highly-scalable dragonfly topology. *ACM SIGARCH Computer Architecture News* 36, 3 (2008), 77–88.

[21] Christopher Kleman, Shoaib Anwar, Zhengchun Liu, Jiaqi Gong, Xishi Zhu, Austin Yunker, Rajkumar Kettimuthu, and Jiaze He. 2023. Full Waveform Inversion-Based Ultrasound Computed Tomography Acceleration Using Two-Dimensional Convolutional Neural Networks. *Journal of Nondestructive Evaluation, Diagnostics and Prognostics of Engineering Systems* 6, 4 (2023), 041004.

[22] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. 2020. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems* 2 (2020), 230–246.

[23] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[24] Zhengchun Liu, Tekin Bicer, Rajkumar Kettimuthu, and Ian Foster. 2019. Deep learning accelerated light source experiments. In *IEEE/ACM Third Workshop on Deep Learning on Supercomputers*. IEEE, 20–28.

[25] Zhengchun Liu, Rajkumar Kettimuthu, Michael E Papka, and Ian Foster. 2023. FreeTrain: A Framework to Utilize Unused Supercomputer Nodes for Training Neural Networks. In *IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 299–310.

[26] Zhengchun Liu, Hemant Sharma, Jun-Sang Park, Peter Kenesei, Jonathan Almer, Rajkumar Kettimuthu, and Ian Foster. 2020. BraggNN: Fast X-ray Bragg Peak Analysis Using Deep Learning. *arXiv preprint arXiv:2008.08198* (2020).

[27] Ahuva W. Mu'alem and Dror G. Feitelson. 2001. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems* 12, 6 (2001), 529–543.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703* (2019).

[29] Tirthak Patel, Zhengchun Liu, Rajkumar Kettimuthu, Paul Rich, William Allcock, and Devesh Tiwari. 2020. Job Characteristics on Large-Scale Systems: Long-Term Analysis, Quantification and Implications. In *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 1186–1202.

[30] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R Ganger, and Eric P Xing. 2021. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning.. In *OSDI*, Vol. 21. 1–18.

[31] Aurick Qiao, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R Ganger, and Eric P Xing. 2020. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. *arXiv preprint arXiv:2008.12260* (2020).

[32] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, Vol. 33. 4780–4789.

[33] Alexander Sergeev and Mike Del Balso. 2018. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[34] Sathish S Vadhiyar and Jack J Dongarra. 2003. SRS: A framework for developing malleable and migratable parallel applications for distributed systems. *Parallel Processing Letters* 13, 02 (2003), 291–312.

[35] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. 2019. NAS-Bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*. PMLR, 7105–7114.

[36] Andy B Yoo, Morris A Jette, and Mark Grondona. 2003. Slurm: Simple Linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*. Springer, 44–60.

[37] Haihang You and Hao Zhang. 2012. Comprehensive workload analysis and modeling of a petascale supercomputer. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 253–271.

[38] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. 2018. ImageNet training in minutes. In *47th International Conference on Parallel Processing*. 1–10.

[39] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.