Cross-System Analysis of Job Characterization and Scheduling in Large-Scale Computing Clusters

Di Zhang[1], Monish Soundar Raj[1], Bing Xie[2], Sheng Di[3], and Dong Dai[1]

[1]University of North Carolina at Charlotte, USA, {dzhang16, msoundar, ddai}@charlotte.edu
[2]Microsoft, USA, bingxie@microsoft.com
[3]Argonne National Laboratory, USA, sdi1@anl.gov

***Abstract*—Amid the growing prevalence of artificial intelligence (AI) and deep learning (DL) across industries and science disciplines, high-performance computing (HPC) clusters are increasingly used for DL tasks, in addition to their traditional role in numerical simulations. This shared use of HPC systems for both DL tasks and numerical codes is altering the characteristics of their workloads, leaving many previously observed and well-accepted workload characteristics unchecked, potentially outdated and imprecise, for these new mixed workloads. Thus, to understand these changes and their implications for job scheduling, we conduct a cross-system analysis of job characterization and its scheduling across a range of representative clusters, including two classic HPC clusters (Mira, Theta), two classic DL clusters (Philly, Helios), and a hybrid cluster (Blue Waters).**

Our cross-system analyses focus on three key aspects: 1) job geometries (job size, run time, arrival interval) and their impacts on scheduling results, 2) job failure patterns and their correlations to job geometries, 3) per-user behaviors and their indication on job scheduling. Through these comparisons, we confirm notable disparity and similarity among different systems (summarized as 8 takeaways), which would help design more efficient job schedulers for the future HPC systems. We further introduce two use case studies (*job runtime prediction* and *adaptive relaxed backfilling*) that leverage these new observations and show the improved job scheduling results. In summary, we expect our observations, insights, and systematic analysis approach can be useful for the community in building efficient HPC scheduling for the upcoming hybrid workloads.

I. Introduction

High-performance computing (HPC) systems were traditionally designed for rigid and long-term scientific simulations and analyses. Recently, there has been a notable adoption of deep learning (DL) across science domains, such as climate modeling [22], molecular simulations [19], fusion simulations [9]. Compared to traditional HPC workloads, these DL-based applications are known to exhibit vastly different characteristics, such as inherent heterogeneity (leveraging both CPUs and GPUs) and feedback-driven exploration (high cancellation rate) [17]. The increasing popularity of DL workloads in HPC computing clusters is changing the workload characteristics, potentially impacting their job scheduling designs.

Workload characterizations have been proven effective for gaining insights into large-scale systems and their job scheduling. There have been a large number of studies on characterizing various HPC workloads [31], [29], [8], [44], [2], [20], [32], [43], [33]. Unfortunately, existing studies do not consider the characteristics of emerging DL workloads, simply because the job traces they analyzed (typically before 2017) do not contain the emerging DL workloads yet. Consequently, many of their identified workload characteristics might not hold true any more, while, important characteristics may preserve at the same time. Understanding them will be critically important for improving future job scheduling designs.

Similarly, there have been recent studies analyzing dedicated DL workloads and proposing new job schedulers accordingly, such as Tiresias, Gandiva, and Optimus [17], [23], [47], [14], [30]. However, they target only industrial DL clusters, providing minimal information to the HPC community and offer limited insight for HPC job scheduling. In addition, none of these works conducts a cross-system analysis to understand how such new workloads could change HPC job characteristics and impact its scheduling.

Different from existing studies, this paper conducts a cross-system comparative study trying to understand the changed and unchanged characteristics while DL workloads are emerging in traditional HPC computing clusters. Specifically, we rely on public job traces collected from five distinct computing clusters to conduct the cross-system analyses, including two classic HPC clusters (Mira and Theta), two classic DL clusters (Philly and Helios), and a hybrid cluster (Blue Waters). The reason of selecting these job traces for analysis will be discussed in §II.

Across target systems, we first study their key job geometries, such as average job sizes, run time, submission interval, and their impacts on scheduling results (e.g., waiting time, system utilization). Next, we study the job failure patterns across workloads and demonstrate the similarities on job failures in both classic HPC and DL workloads. Thirdly, we focus on per-user behaviors by profiling the user behaviors across the target systems and identifying consistent patterns and behaviors. Finally, built upon the observations derived from our analyses, we introduce two case studies (*job run time prediction* and *relaxed backfilling*) to improve the efficiency of HPC job schedulers. We evaluate and verify the effectiveness of the optimizations through extensive simulations. Our key contributions of this study can be summarized below:

- To the best of our knowledge, we conduct the first systematic job characteristic comparisons across HPC and DL workloads based on job traces collected from

both leadership supercomputers and leading industry DL-centric data centers. We accordingly derive ***eight*** major takeaways regarding the job geometrics, job failure patterns, and user behaviors, which will help optimize job scheduling for hybrid workloads.
- To showcase the applicability of our key observations, we conduct ***two*** use case studies to improve existing HPC job scheduling mechanisms or components and show their effectiveness. Specifically, we introduce 1) elapse-time based job runtime prediction and 2) adaptive relaxed backfilling. We conduct extensive simulation-based evaluations and report up to 15% reduction in bounded job slow down and 49% reduction in job delay, showing the effectiveness of our observations.

Limitation. In addition to the new observations and findings, we acknowledge the constraints inherent in this study. Despite we conducted comparisons using the traces from multiple state-of-the-art DL clusters, it is still possible that our observations may not entirely capture the nuances of the latest DL workloads given the rapid evolution of DL models.

To address this issue, we summarize our analytic methodology as a software package[1] and a website[2] for others to easily conduct similar analysis using their own job traces and compare with the systems discussed in this study. We expect our proposals of the systematic methodology and the metrics can be applicable to any other job traces, hence generally useful for the community.

This paper is organized as follows: In section §II we introduce the job traces used in this study, how we process them, and our analysis approaches. In sections §III, §IV, §V, we present the cross-system analyses and comparisons from three main aspects: job geometries, job failures behaviors, and users' behaviors. Based on these observations, we present two use cases to improve HPC job schedulers in §VI. We compare with related work in §VII, conclude this paper and discuss the future work in §VIII.

II. Data Preparation and Analysis Methodology

A. Job Traces Selection and System Descriptions

We collected 10 publicly available job traces from real-world computing clusters for cross-system analysis, as summarized in Table I. These traces include those from HPC clusters operated in academic or national labs, such as Mira [5], Theta [38], ThetaGPU [39], Blue Waters [27], and Supercloud [34], as well as traces from industry data centers such as Philly [18], Elasticflow [13], Helios [17], and Alibaba Cluster Trace [46]. Here, we focus only on job traces that are later than 2017 to gain the latest trend.

Among them, we have to exclude traces that are not suitable for our analysis. For example, ThetaGPU only has 24 nodes, so it does not represent a large cluster. Elasticflow and Alibaba Cluster Trace were also left out because they have too few jobs, affecting analysis conclusions. Some of them even miss key metadata such as user information or job status information. Supercloud met these requirements, but we noticed some inconsistent information while analyzing it. For instance, it reported a total node count of 704, but the trace showed many jobs with requested nodes exceeding 704 being successfully scheduled. Therefore, we decided not to include it until gaining more accurate understanding of its information.

The selected target systems include Mira, Theta, Blue Waters, Philly, and Helios. Among them, Mira and Theta are supercomputer housed at the Argonne Leadership Computing Facility (ALCF) [5], [37]. Mira consists of 49,152 compute nodes, each equipped with 16 CPUs. At one point, Mira held the 3rd place on the TOP500 list of HPC clusters. Theta comprises 4,392 compute nodes, each equipped with 64 CPUs. Theta ranked 94th on the most recent Top500 list. ALCF periodically released its job traces, which had inspired numerous studies on workload characterization and scheduling [29], [8], [1], [48]. Since Mira and Theta mainly consist of CPUs to run traditional HPC applications, such as HOOMD simulation from materials science [6], we consider Mira and Theta represent traditional HPC workloads.

Comparatively, housed at the National Center for Supercomputing Applications (NCSA) [27], Blue Waters represents a hybrid architecture, which consists of 22,636 CPU nodes and 4,228 GPU nodes to support both traditional HPC and new machine learning applications. It is likely that future HPC clusters will adopt a similar heterogeneous architecture with both CPUs and GPUs to support various applications. As evidence, four out of the top five HPC systems currently employ a substantial number of GPUs to accelerate tasks related to AI and DL workloads [40].

Philly and Helios are clusters specifically designed for running industrial DL workloads, providing meaningful insights regarding DL workloads. In particular, Philly [18] is a Microsoft data center built explicitly for DL training around 2017 Helios [17] is a newer data center operated by SenseTime. It began operations in 2020. It was designed for the development of deep learning models for both research and production purposes. Compared with Philly, Helios jobs focus on larger scales: they required over twice the GPUs than that of Phillys. Its maximum number of requested GPUs is 2048, which is an order of magnitude higher than Philly. We include both traces to thoroughly understand various DL workloads.

B. Dataset Alignment

All job traces used in this study are publicly available, but with varying levels of details. For example, the Blue Waters trace includes job scheduling statistics, I/O data, memory allocation states, CPU and GPU usage, and data usage, etc. Philly trace, on the other hand, only includes CPU/GPU utilization, job failures, and the root causes. In this study, we focus on job geometries, job failures, and user behaviors. To conduct cross-system comparisons, we focus on the common attributes and eliminate those specific for certain systems.

In addition to leveraging the aforementioned features, we also align the timeframes among these datasets. As shown

[1]https://github.com/DIR-LAB/lumos

[2]https://lumos-job-traces.streamlit.app/

Dataset	Affliation	Years	Job Count	Nodes	Cores	GPUs	Large Scale	User Info.	Job Status	Info Consis.
Mira [5]	ALCF	2013∼2019	750,000	49,152	786,432	NA	✓	✓	✓	✓
Theta [38]	ALCF	2017∼2023	522,858	4,392	281,088	NA	✓	✓	✓	✓
Blue Waters [27]	NCSA	2013∼2019	10.5 M	26,864	396,000	4,228	✓	✓	✓	✓
ThetaGPU [39]	ALCF	2020∼2023	135,975	24	NA	192	✗ *(Cluster Size)*	✓	✓	✓
Supercloud [34]	MIT	2021-01∼2021-10	395,914	704	32,000	448	✓	✓	✓	✗
Philly [18]	Microsoft	2017-08∼2017-12	117,325	552	NA	2,490	✓	✓	✓	✓
Helios [17]	Sensetime	2020-04∼2020-09	3.3 M	802	NA	6,416	✓	✓	✓	✓
Elasticflow [13]	Microsoft	2021-03∼2021-05	69,351	NA	NA	NA	✗ *(Job Count)*	✗	✗	✓
Alibaba Cluster Trace [46]	Alibaba	2023	8,152	1,523	107,018	6,212	✗ *(Job Count)*	✓	✓	✓

TABLE I: Overview of available and selected public job traces of large-scale computing systems. *N/A indicates no corresponding information.*

in Table I, some traces, such as Mira and Theta, cover a long time period, spanning from 2013 to 2019. While, other traces, such as Philly and Helios only obtain the trace data for several months in 2017 and 2020 respectively. It is hard to obtain a fair comparison between the workloads of ten years and those covering about four months. To address this issue and conduct a meaningful cross-system analysis and future workloads projection, we choose to analyze the latest four-months traces from Blue Waters (2019-08∼2019-12), Theta (2022-12∼2023-05), and Mira (2019-08∼2019-12).

C. Terminology and Analysis Approaches

In the context of job scheduling, *Users* submit *Jobs* to computing systems and clusters, where each job is a single execution instance that utilizes one or more compute nodes (*Job Size*) and executes for a period of time (*Run Time*). A *Job Scheduler*, such as SLURM [49] or YARN [42], determines which jobs to be scheduled next from a set of jobs in the *waiting queue*. The time duration each job spends in the waiting queue is referred to as *Waiting Time*.

HPC job schedulers make scheduling decisions with various strategies, such as First-Come-First-Serve (FCFS) or Shortest Job First (SJF). To improve scheduling effectiveness, these schedulers often leverage the *Backfilling* mechanism [26], [24], which opportunistically fills idle resources even when they are reserved for another job. The performance of job scheduling is evaluated by metrics such as *System Utilization (util)*, which calculates the ratio between utilized core hours v.s the total core hours of the system, *Average Waiting Time (wait)*, which calculates the average job waiting time in the queue, or *Slowdown* which means the ratio of job turnaround time over its execution time. The *Average Bounded Slowdown (bsld)* [12] measures job slowdown relative to given "interactive thresholds" (e.g., 10 seconds).

In addition to statistical analyses, we also leverage a widely used HPC job scheduling simulator named SchedGym [50], [51], [21] to schedule the exact job traces using different scheduling strategies and optimizations to examine how job characteristics impact scheduling strategies,

III. Job Geometries and Scheduling

The first set of comparative studies focus on job geometries and their connections with job scheduling results.

A. Job Geometries Comparisons

we first analyze the following three geometries: 1) *Job Run Time*, 2) *Job Arrival Patterns*, 3) *Job Resource Allocation*. To drive a set of systematic comparisons on the similarities and differences, we conduct the same analysis across all job traces. The overall results are summarized in Fig 1.

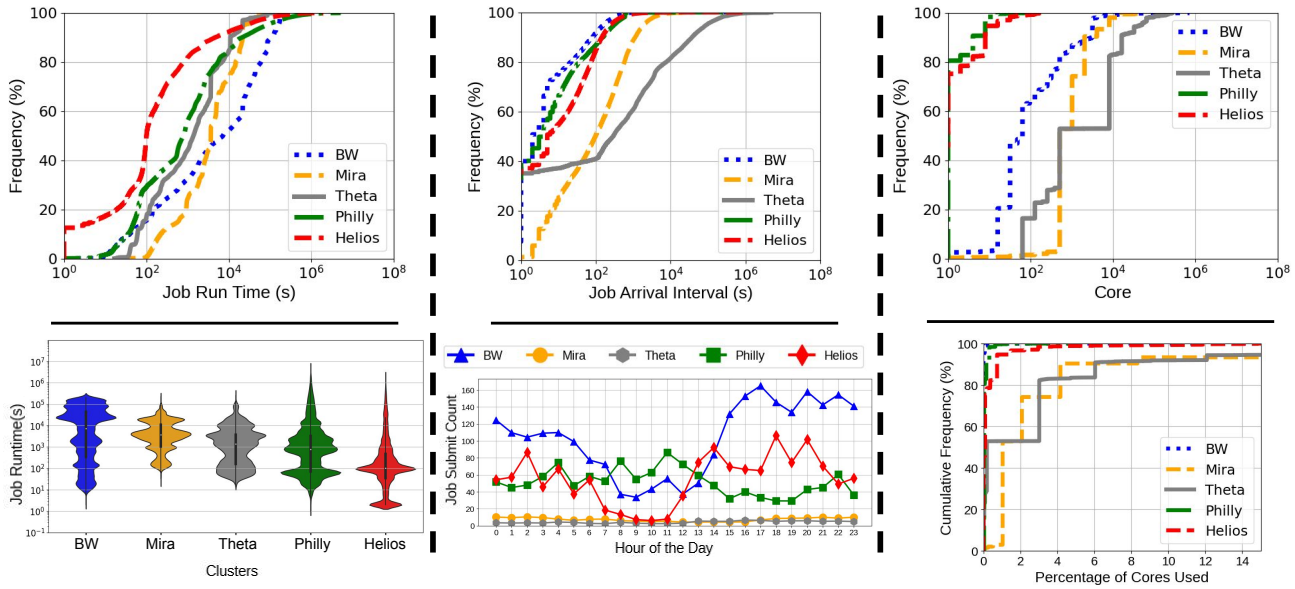
Job Runtime. Figure 1(a) upper presents the Cumulative Distribution Functions (CDFs) of the job runtime of the target systems. The plot illustrates the disparity among them: the jobs in Mira, Blue Waters, and Theta run longer than those in the deep learning (DL) systems (Philly and Helios) do. For instance, the median runtime on Blue Waters and Mira are approximately 1.5 hours, while the median run times on Helios and Philly are 90 seconds and 12 minutes, respectively. we would expect the schedulers to focus more on minimizing job waiting times while more DL workloads are emerging.

We further plot the violin distributions of job runtime of different systems in Figure 1(a) bottom to help understand their detailed differences. One key observation here is that the traditional HPC clusters, such as Mira and Theta, have relatively stable job run times, while DL clusters (Philly, Helios) have more diverse job run times. Their longest job runtime could be significantly longer than that of traditional HPC systems. Their shortest job runtime is also much shorter. Blue Waters, as a hybrid system, exhibits a middle ground between these two extremes. We suspect these extremely long jobs from Philly and Helios would be DL training workloads that could take weeks and months to finish [7], [3], [35].

Takeaway 1:

The shorter and more diverse job run time in DL clusters suggests that the future job scheduling might need to rethink their key designs relevant to job run time. For example, many studies use average bounded job slow down [12] to measure the scheduling policies. It sets the lower bound at 10 seconds with the expectation that such short jobs are unlikely. However, looking ahead, these short jobs might be more common, prompting the need for a reconsideration of such an important metric.

Job Arrival Patterns. Figure 1(b) upper presents the job arrival patterns of different systems, which have a profound impact on job scheduling [36]. We can observe that the job



(a) Job Runtime (b) Job Arrival Pattern (c) Job Resource Allocation

Fig. 1: Job geometries (job runtime, job arrival interval, and job resource allocation) characterization of multiple workloads.

arrival intervals are much smaller in the hybrid or DL-based clusters, such as Philly, Helios, and Blue Waters. In fact, over 50% of their jobs arrive within 5-10 seconds after the previous jobs. For traditional HPC clusters and workloads, such as Mira and Theta, this number is 10 times larger (100 seconds), demonstrating a much lower frequency. We will discuss how this would impact the job scheduling in later subsections.

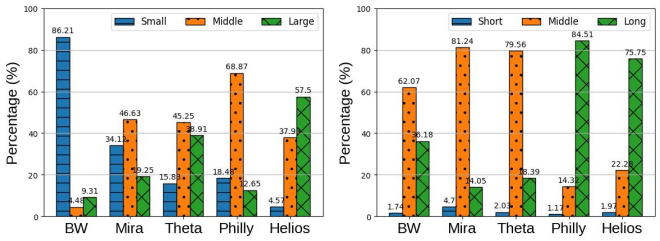
In addition to job arrival frequency, we further show the cyclic patterns of job arrivals in Figure 1(b) bottom, which plots the job arrival counts for each hour of the day (hour from 0 to 23). Theoretically, job arrival is often considered as a Poisson process. However, in practice, it often exhibits periodic patterns. We examine whether these periodic patterns are consistent across different workloads and clusters. Note that, since different clusters may be in different time zones (e.g., Mira and Theta are in Central Time and Philly is in Pacific Time), we always use their local time in this plot.

From these results, we first observe an increase in the number of submitted jobs from 8am to 5pm in many of the systems. This is often summarized as the 'peak hours' pattern and leveraged to autoscale the system in many existing studies [25]. However, such a pattern does not exist in Mira, Theta, and Philly. First, in these workloads and throughout the day, there is no notable favorable time for job submissions. For instance, the lowest number of job submissions in an hour in Philly is around 40, and comparatively the maximum is less than 100 (2.5x difference), which is much smaller than the same max-min ratio of Helios (10 and 110, 10x difference). Secondly, these workloads follow a different time pattern, where Mira and Theta receives slightly more jobs after 12pm, while Philly receives fewer jobs during the 'peak hours'. These differences could originate from diverse users working with different habits or simply from different time zones.

Takeaway 2:

These observations show that although the periodic patterns still exist in many workloads, its generality is not certain, i.e., the intensity and time can be vastly different, suggesting that job scheduling should be cautious about leveraging the periodic patterns for general purpose. The periodic pattern should be identified per system and used only for that system.

Job Resource Allocation. Figure 1(c) upper shows the job resources requirements. It plots the CDF of requested cores per job. Here, for Philly and Helios, the requested cores are GPUs; for Mira and Theta, the requested cores are CPUs; for Blue Waters, they are hybrid. From the plot, we can clearly observe the divided nature of the target systems: Philly and Helios belong to one category; Mira and Theta belong to another category; Blue Waters is in the middle. Specifically, we can observe that about 80% of GPU-based DL workloads (Philly, Helios) require only a single GPU core, significantly smaller than the number of CPU cores a traditional HPC job requests. In fact, more than 50% of Mira jobs request more than 1,000 cores, substantially larger than DL-based GPU systems. Blue Waters takes a middle ground. The median number of requested nodes for Blue Waters is merely 32, which is still larger than classic DL workloads. In fact, around 90% of Blue Waters jobs request more than 10 cores. In addition to show the absolute value of allocated cores, we further plot the percentage in Figure 1(c) bottom. The results follow similar divided patterns across systems. The only difference is the percentage of Blue Waters is much smaller.



(a) Core Hours w.r.t Job Size (b) Core Hours w.r.t Job Run Time

Fig. 2: Core hour domination of different types of jobs.

Takeaway 3:

Looking ahead, more small (and short) jobs are expected in computational clusters, which may impact job scheduling optimizations. In fact, leveraging this new job characteristic, we identify opportunities to improve job scheduling efficiency with relaxed backfilling. We visit this topic in Section VI.

Resources Usage v.s. Job Size and Run Time. We further study how resources are used by different types of jobs. In particular, we categorize jobs into three sizes (small, middle, and large) based on the job size (number of cores they use) and three lengths (short, middle, long) based on job runtime (how long they run) and study the core hours they use.

For Mira and Blue Waters, we define small, middle and large by following the conventional numbers used in previous studies [29], where small represents a job allocates less than 10% of total cores, middle means the job size is in the range of 10% - 30%, and large means the job size is >30%. For DL workloads, we define job sizes differently as the majority of them request less than 10% of the total GPUs. Following the conventional numbers used in the DL workload studies [17], we define a job as small if 1 GPU in use, middle if 1 to 8 GPUs in use, and large when >8 GPUs in use. For job runtime. We follow the same categorization rule [31] for all four traces: short (<1 hour), middle (1 hour to 1 day), and long (>1 day).

Figure 2 plots the results. We can observe that the dominating groups of jobs change across different systems. For example, small jobs in Blue Waters account for more than 85% of the total core hours, while small jobs in Mira, Theta, Philly, and Helios consume less than 35%, 16%, 19%, and 5% of total core hours, respectively. Their dominating job sizes are different as well. For job runtime, surprisingly, different from the intuitive thought that long-running jobs would consume the largest percentage of core hours, we observe that Blue Waters, Mira, and Theta are dominated by middle length jobs. Philly and Helios are dominated by long jobs. It is interesting to see that classic HPC workloads exhibit a clear bias toward middle length jobs, but DL workloads exhibit a greater bias towards long jobs, indicating a strong shift.

Takeaway 4:

Across systems, dominating job groups (taking more than 50% of the whole system core hours) widely exist but shift. It is then critical to identify these groups and optimize the scheduling policies accordingly, instead of only focusing on large jobs as traditional HPC schedulers did [1].

B. Job Scheduling Results Analysis

We then see how these job geometries would impact the final job scheduling results across systems. We analyze two key job scheduling metrics: *System Utilization* and *Average Job Waiting Time*, which were discussed in Section II.

System Utilization. Fig 3 displays the system utilization of the target systems. Since Blue Waters obtains both CPUs and GPUs, we plot the CPU and GPU utilization separately. Although Philly and Helios obtain both CPUs and GPUs, their CPUs primarily serve as auxiliary resources. The traces do not mention the scale of CPUs either. Therefore, for Philly and Helios, we only plot GPU utilization.

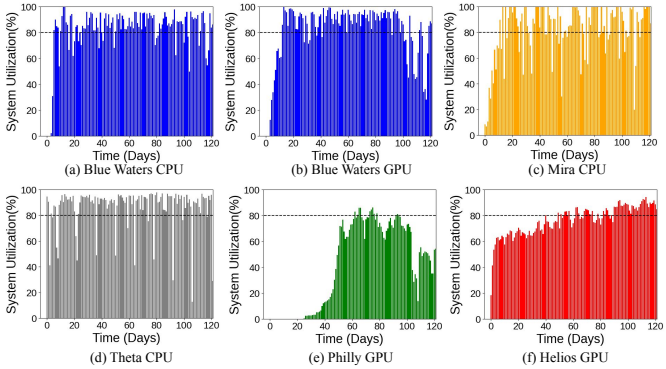


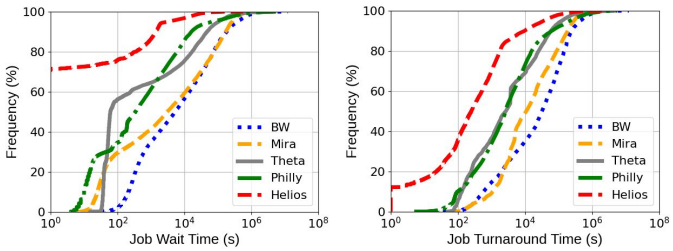
Fig. 3: The system utilization across multiple systems.

From the results, we observe that both Philly and Helios exhibit much lower system utilization than the other systems. Most of the time, less than 80% of the GPUs are used, even when there are jobs waiting in the queue (see the later job waiting time analysis). Given most of the jobs in these systems request a small number of GPUs (as shown in Fig. 1(c)), it is counter-intuitive to see these many idle GPUs while there are jobs waiting in the queue. Looking deeper into the job traces, we noticed one of the major reason would be the isolated virtual clusters created in DL clusters for resource sharing among users and groups. For example, there are 14 virtual clusters in Philly. Then, a job will be queued in each virtual cluster until its requested GPUs are available in the same virtual cluster. This isolation mechanism is designed to avoid interference among jobs from different users and groups, but leads to extra waiting even when there are available resources in the system. Note that, the extremely low system utilization in Philly (43% average utilization) might be relevant with the long vacancy period at the beginning phase of the Philly trace. But, even without considering this beginning phase, its utilization is still low and barely over 80%.

Takeaway 5:

The low utilization in DL clusters highlights their issues and opportunities of job scheduling policies when facing the short but high variance workloads, possibly created by deep learning long training tasks and short inference tasks.

Job Waiting Time. Figure 4 compares the CDF of job waiting time and job turnaround time across target systems. From these results, we have several observations. First, among all systems, Helios shows the minimal job waiting time: about 80% of the jobs wait less than 10 seconds. In contrast, for the same DL-based Philly, the number is much larger: over 50% of the jobs need to wait at least 10 minutes to start. Considering the low system utilization of the Philly cluster discussed earlier, the scheduling policy used in Philly clearly encountered issues. Its fair-sharing scheduling policy is not working optimally when dealing with isolated virtual clusters and unbalanced job submissions from different users. Looking into the actual scheduling trace, we do often find jobs are waiting on one virtual cluster while other virtual clusters are idle.



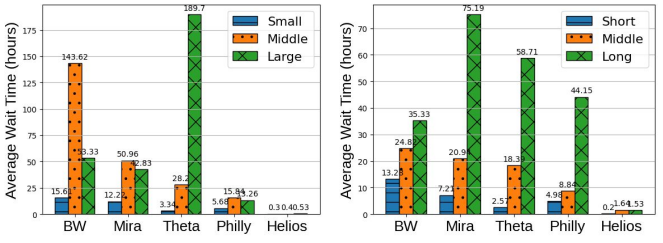
(a) CDF of Wait Time (b) CDF of Turnaround Time

Fig. 4: Comparisons of waiting time and turnaround time.

Second, it is interesting to observe that Blue Waters has the longest average job waiting time: over 50% of the jobs need to wait more than 1.5 hours, which is roughly equal to its own average job runtime (1.5 hours). Looking at the job geometries of Blue Waters in Figure 1, we can see such a long waiting time should result from its long average job run time and small (frequent) job arrival interval requested by each job. Across target systems, Mira shares similar job geometries with Blue Waters, but its average job waiting time is obviously shorter. We believe it is the hybrid workloads of Blue Waters, a mix of extremely long and short jobs (as shown in Fig. 1(a)) that create significant challenges for its scheduler. This also calls for substantial improvements over HPC job schedulers to prepare for future workloads. Similarly, we also show the CDF of job turnaround time, which is the sum of job waiting time and job runtime, in Figure 4(b). The results follow similar patterns as the waiting time.

Figure 5 further shows the relationship between the observed job waiting time and job geometries across different systems. We use the same way (described previously) to group jobs into 'small', 'middle', and 'large' based on job size, and 'short', 'middle', and 'long' based on the job run time.

There are several interesting observations. First, surprisingly, across most of the systems (except Theta), the middle



(a) Average Waiting Time w.r.t Job Size (b) Average Waiting Time w.r.t Job Run Time

Fig. 5: Correlation between job waiting time and two other geometries: *job size and job run time*.

size jobs, instead of the largest jobs, wait the longest time in queue. This counters the intuitive that larger jobs will always wait longer [31], but actually makes sense: in many production systems, large jobs are often the special-purpose jobs and hence treated specially in schedulers. They often enjoy higher priority and hence less waiting time. Second, unsurprisingly, across all the systems, the long jobs seem to wait the longest time in queue. Echoing findings from studies [10], [1]. We believe this is mostly due to the use of Backfilling that favors shorter jobs during scheduling [24], [26], [41].

Takeaway 6:

The different job waiting time of Philly and Helios shows the importance of cluster management and scheduling. The long waiting time of Blue Waters further highlights the challenges of scheduling hybrid workloads. All motivate the need for better scheduling policies for future workloads in HPC.

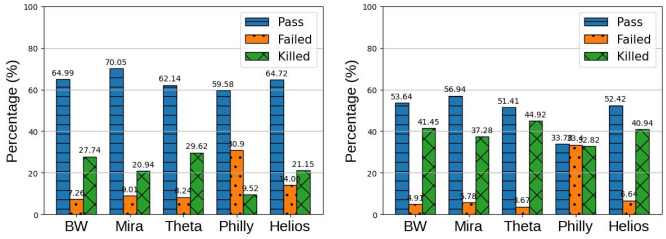
IV. Job Failure Characterization and Scheduling

It is often a myth that HPC jobs appear to be more stable and less likely to fail, while DL jobs exhibit a higher failure rate. It is commonly believed the higher failure rate of DL workloads comes from the higher cancellation rate due to model and hyper-parameters explorations. In this section, we systematically examine these assumptions and how they may impact HPC job scheduling.

A. Job Failures Distribution

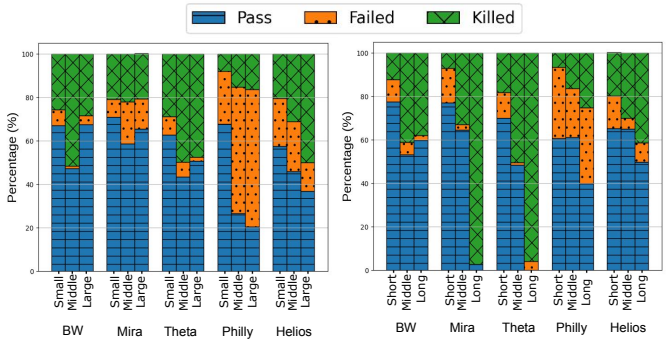
We define **Job Status** based on its final exit status printed in job traces. We focus on three possible statuses: Passed (job finishes normally), Failed (job fails in the middle due to technical issues), and Killed (job is killed by external factors before finishes). These three statuses are originally provided in Philly and Helios traces, but not given in other traces. In these systems, detailed job exit information, such as SIGTERM, SIGKILL, and SIGABRT, is provided, which can be used to classify jobs into one of these three statuses. Specifically, since SIGTERM is to politely ask a program to terminate and SIGKILL is to ask to terminate a program, both of them are considered as "Killed". SIGABRT is due to abort assertion or double-free of memory and SIGSEGV is segmentation fault, we classified them as "Failed".

Based on these definitions, we show job counts percentages and consumed core hours percentages of different jobs statuses



(a) Job Counts w.r.t Job Status (b) Core Hours w.r.t Job Status

Fig. 6: The distribution of different job statuses.



(a) Job Status w.r.t Job Size (b) Job Status w.r.t Job Run Time

Fig. 7: Job failure v.s job runtime and job requested resources.

in Figure 6. From these results, we have several interesting observations. First, the percentages of 'Passed jobs' are less than 70% across all the systems. Even in Mira, a mature HPC system, the failed or killed jobs still take around 30% of the total number of jobs. There is no significant difference between classic HPC and new DL workloads in terms of job failure rate: Philly has the highest failure rate at 40% v.s the lowest failure rate at 30% in Mira. Second, the core hours consumed by the killed jobs take significantly higher percentages compared with their job numbers, indicating that 'unfinished' jobs actually waste more resources. For instance, in Philly, nearly 60% of the jobs are passed, but they only consume 34% of the GPU resources. The rest 66% of the GPU resources are wasted on either Failed or Killed jobs. Third, in most of the systems, there are more number of 'Failed' jobs than their corresponding consumed core hours (e.g., 7.3% v. 4.9% in Blue Waters), indicating these jobs cost less. One explanation is that Failed jobs are often caused by software bugs or wrong configuration settings, hence likely 'fail' at a very early stage of their executions. From these observations, it is not hard to conclude that the 'Killed' jobs are an important issue to address in large-scale computing clusters, as they widely exist and consume a significant amount of resources.

B. Correlation between Job Failure and Job Geometries

We take a closer look at the correlation between job failures and job geometries, particularly job run time and requested resources. We follow the same way to categorize jobs into *small, middle, and large* jobs based on job sizes; *short, middle, and long* jobs based on job runtime.

Figure 7(a) shows how is the job failure rate correlated with job sizes. In each bar, we show the percentages taken by Passed, Failed, and Killed jobs within that job size category. It is interesting to see the percentages of Passed jobs in Philly and Helios reduces significantly as job size increases. But, such an observation does not hold in Blue Waters, Theta, or Mira, where the job size seems irrelevant with the job statuses.

Figure 7(b) plots how is the job failure rate correlated with job run time. It is obvious that the job run time impacts the distributions of the job statuses significantly in all systems. The number of Passed jobs decreases as the jobs become longer. In some extreme cases, such as Mira, almost 99% of the Long job will be killed eventually. The reduction of Passed jobs mostly comes from more Killed jobs.

Takeaway 7:

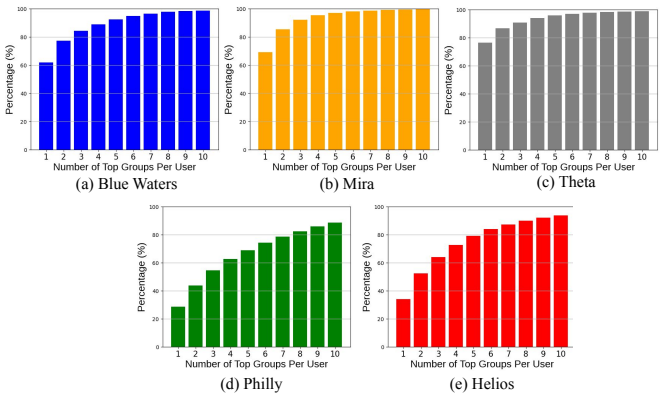
The consistent high job failure rate and high costs of failed or killed jobs across systems suggest a great need to manage job failures proactively in job scheduling. Many of the previous fault-aware schedulers [28], [52], [37] should be revisited in the new hybrid workload setting.

V. User Behaviors and Scheduling

In this subsection, we further extend our characterization to more users' behaviors with the goal of identifying common patterns that can be effectively leveraged in job scheduling.

A. Users' Repeated Behaviors

It is previously identified that HPC users will likely repeatedly submit jobs needing similar resources and executing similar amount of time [29]. To verify whether this also applies to DL-mixed workloads, we follow the same approach of the previous work to analyze the "resource-configurations" metric of jobs [29]. The resource-configurations metric is a pair of "[cores, run time]". We first calculate the paired values for each job, then group all the jobs that have "similar" job resource-configuration. For two jobs from a user to belong to the same group, they have to have exactly the same number of nodes and run times within 10% of their mean run time.



(a) Blue Waters (b) Mira (c) Theta (d) Philly (e) Helios

Fig. 8: The resource-configuration group per user.

Figure 8 shows the percentage of jobs that are in the top 10 largest group per user among all the submitted jobs from

that user. These are averages across representative users. We incrementally plot the percentage, meaning group 2 includes jobs from both groups 1 and 2. The results shows, across all systems, the jobs are highly repeated per user. Nearly 90% of the jobs submitted by the users belong to the first 10 groups (among over hundreds of possible groups). There are still small differences though. if we only focus on the first 3 groups, we will find DL workloads (Philly and Helios) barely reach 60%, while Blue Waters, Mira, and Theta already pass 80%, showing less repeated patterns for DL workloads.

B. Users' Submission Behaviors

The rational behind the users' repeated behaviors is that each user has a set of applications to run, so these submitted jobs follow certain patterns statistically. However, in a fine-grained aspect, users do have flexibility each time when they submit the job. For example, they may determine how many resources to request based on the system's availability. Or they may change their work schedules, so that they can submit large jobs when the cluster is less busy. We consider such users' behaviors important to examine whether they are consistent across systems.

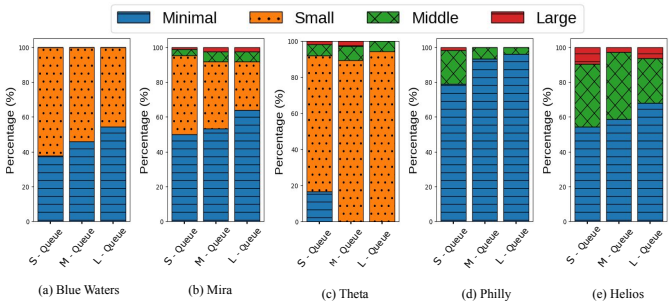


Fig. 9: Submitted jobs' sizes impacted by queue length.

First, we show how users might change their behaviors in requesting resources facing different job queue length of the system. Specifically, we iterate each job submission event and record the requested resources of the job and the queue length at that time. We consider three different queue lengths (short, middle, and long) for each system as Fig. 9 shows. For each system, we consider the maximal queue length as Q, then define 'short' queue as cases whose queue length is less than 1/3*Q, 'middle' between 1/3 and 2/3*Q, and 'long' longer than 2/3*Q. We tried different fine-grained queue classifications and observed similar results. Then, each bar in Fig. 9 represents one queue length. We show the percentage of jobs requesting 'small', 'middle', and 'large' resources within each bar. The classification is the same as previous sections. We do add one 'Minimal' category which means only requesting one CPU or GPU. Having this category helps showcase the trend in traces like Blue Waters and Mira, where 99% of the jobs are already 'small' regardless of queue length.

The results show a clear trend across most of the system: as queue length increases, users tend to submit jobs needing less resources. For instance, in Philly, when the queue length is longer than 3,000 jobs, almost 100% of the jobs are requesting

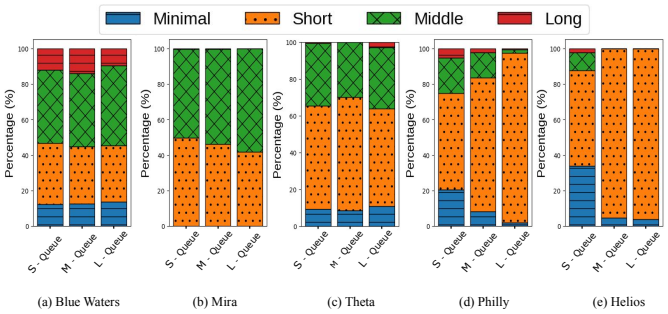


Fig. 10: Submitted jobs' runtime impacted by queue length.

only 1 GPU. But when queue length is smaller than 1,000, the ratio drops to less than 80%. Similar behaviors can be observed in other three systems as well. This is reasonable as when a queue experiences congestion, individuals tend to submit jobs with less resource requirements to prioritize their executions.

We further examine whether the system queue length would impact the run time of requested jobs, i.e., whether users will submit shorter jobs when the system is busy and overloaded. Note that, although users may not know the exact runtime of their jobs when submitting them, they do have an estimation of the execution time, which could be a factor impacting their submission behaviors. We define the queue length and job run time the same way as the previous experiments. We also add a 'Minimal' category for job run time to indicate jobs finished within 60 seconds, to better show the trends for traces whose run times are mostly small.

Fig. 10 shows the results. We can observe a consistent trend in DL workloads: users indeed submit shorter jobs when the system is busy. However, such an observation is not true for Mira, Theta, and Blue Waters. In these systems, the job run time is barely impacted by the queue length. We believe this might be because in these systems, the job run time is not a direct factor in determining the probability of its execution. So, users do not consider it during job submissions.

C. Users' Job Failure Behaviors

The previous per-user consistent behaviors inspire us to further investigate the per-user job failure behaviors. Specifically, we are particularly interested in how job runtime might be relevant with different job statuses, such as Pass, Failed, or Killed. Such a relationship may help us more accurately predict job statuses based on current runtime.

In this study, we report the top 3 users, who submit the most number of jobs in each system due to space limits. We examined the top 10 users and observed similar results. We plot the violin distribution of job run time for each user and each job status in Figure 11. The Theta results are similar to Mira's, hence eliminated to save space.

From the figure, we can see that, for different users, the job run time distribution shifts among different job statues significantly. For instance, users *U1, U2, U3* in Blue Waters all experience a much higher run time for Failed and Killed jobs than that of Passed jobs: their widest (high density) parts depart significantly. Such distribution differences could be very

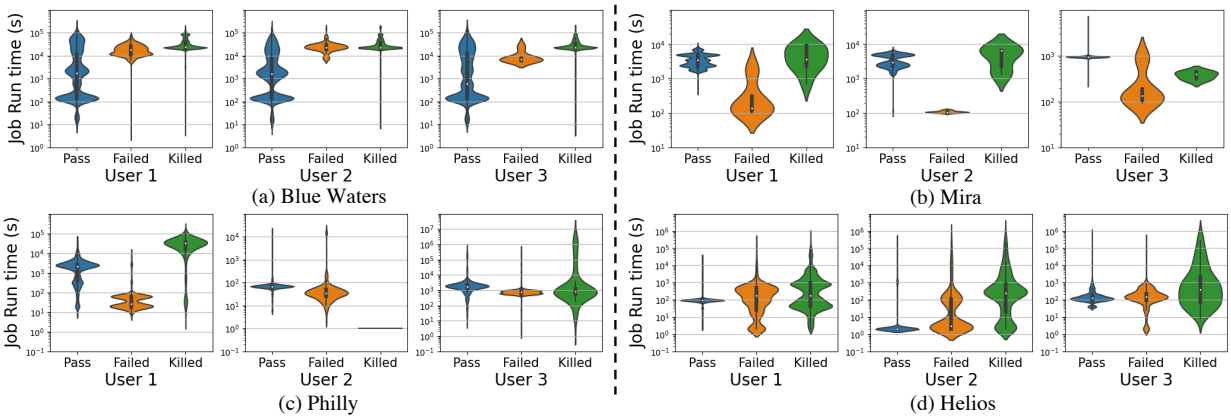


Fig. 11: Per user Job Runtime Distribution v.s. Job Status.

helpful to predict the job status. For instance, for *U1* in Philly, if a job running longer than 10^4 minutes, then it is highly likely to be killed, simply because the possibility for Passed jobs continuing to run over that time is close to 0. Also, if a job runs past 10^2 seconds, then it is likely not gonna Fail. Leveraging such information, schedulers may reversely predict job run time, which is helpful in making effective scheduling decisions. In the next sections, we will show how this can be used to improve the job scheduling performance.

Takeaway 8:

Across systems, we observe consistent per-user patterns in job submissions and job failures, such as when the system is busy, users tend to submit shorter and smaller jobs. It becomes extremely promising to track and analyze users (especially the heavy users) periodically and leverage their behaviors to achieve better scheduling.

VI. Case Studies

In this section, we will showcase two use cases to demonstrate the usefulness of our observations.

A. Use Case 1: Improve Job Run Time Prediction.

In the first use case, we leverage two previous observations:

- The high job failure rate on both HPC and DL workloads means job failures are common and should be effectively considered during scheduling;
- For many users, the job run times are not evenly distributed across job statuses (i.e., Passed, Failed, Killed), resulting in a consistent correlation between them.

We propose to use the elapsed time of a job, i.e., time that has already been consumed, to predict the remaining job runtime.

Specifically, from Fig. 11, we can see that, once a job execution time (from a given user) has surpassed a threshold, then it is highly likely to reach the next threshold rather than complete in the middle of the two consecutive thresholds. To elaborate further, assuming the scenario where for a user, most of the failed jobs are $\leq$10 seconds, and most of its normal jobs execute around an hour. Thus, if a new job from the user has already executed for $>$10 seconds, then with high chances it will successfully complete about an hour later, rather than completing or failing in 30 minutes. It is intuitive that, for a user, when we predict a job completion in consideration of a job's current elapsed time, the prediction accuracy can be significantly improved.

To examine the effectiveness of this simple idea, we incorporate a job's current elapsed time into the state-of-the-art runtime prediction models, such as Last2 [41], Tobit [11], XGBoost [4], Linear Regression (LR) [15], multilayer perceptron (MLP) [16], and compared the prediction accuracy before and after integrating this feature.

We use two popular metrics to measure the improvements in run time predictions [41]. First, we compare the *Prediction Accuracy*, calculated as $\frac{\min(runtime,predict)}{\max(runtime,predict)}$, to understand how closely is the predicted job run time compared with the actual run time. Higher prediction accuracy is better. Second, we compare *Underestimation Rate*, which represents the ratio of the number of underestimated job run times to the total number of jobs in a trace. A smaller value is better. The Underestimation Rate is often a more important metric in runtime prediction, simply because underestimating job run time can have worse consequences [11]. It may cause schedulers to backfill inappropriate jobs, resulting in significant delays. Or the job may be terminated if its actual execution time surpasses the predicted runtime.

In our comparisons, the baseline is the prediction made by the existing method without considering elapsed time, labeled as 'Without Elapsed Time'. We then enhance the baseline methods by incorporating elapsed time to make predictions. To include elapsed time, we must make predictions after the job has run for a certain duration, say 20 seconds. However, if we do this plainly, the baseline methods are not treated fairly, as they may make incorrect predictions on jobs running less than 20 seconds, whereas the 'With Elapsed Time' methods will never make mistakes on jobs running less than 20 seconds. To make a fair comparison, we let all methods make predictions only towards jobs that have been running for a fixed time period, say 20 seconds. The only difference is that 'Without

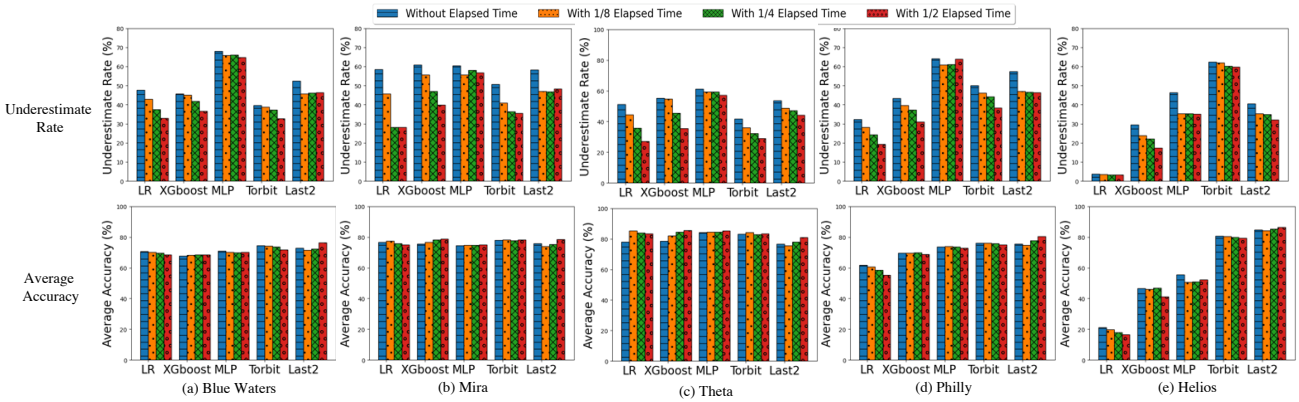


Fig. 12: Job runtime prediction with/without elapsed time. *For Underestimate Rate, smaller is better; for Average Accuracy, higher is better.*

Elapsed Time' methods do not consider elapsed time, while our methods consider that as one input feature of the model.

The results are shown in Figure 12. We report the comparative results about *Underestimate Rate* at the top and *Average Accuracy* at the bottom. Each predictive model has multiple values corresponding to the baseline and the corresponding, improved methods. To understand how the elapsed time helps prediction, we examined the prediction at elapsed time of 1/8, 1/4, and 1/2 of the average job run time. From these results, we can clearly observe that with the elapsed time, almost all predictive models achieve significantly smaller *Underestimate Rate* and comparable or slightly better *Average Accuracy*. With more elapsed time, the improvements are increasing as well.

B. Use Case 2: Improve Relaxed Backfilling.

In the second use case, we leverage another observation about per-user behaviors: users tend to submit jobs with fewer resource requirements and shorter run times when the system is busy (i.e., with a long waiting queue).

We leverage this observation to improve a particular job scheduling mechanism called *relaxed backfilling*. First, backfilling is a widely used method to improve HPC job scheduling. It allows to schedule low priority jobs ahead of high priority jobs, as long as such actions do not delay the starting times of high priority jobs. In reality, the constraint of not affecting the starting of high priority jobs might be too strict and lead to much less backfilling opportunities. Thus, relaxed backfilling was proposed to alleviate such constraints [45]. It allows to delay the starting times of waiting jobs with a threshold, such as 10% or 20% of the expected job waiting time. Relaxed backfilling could lead to an 87% reduction in monthly waiting time [45]. However, the main drawback of relax backfilling is its additional delay of many high-priority jobs. Can we maintain the same job waiting time reduction while significantly reducing the job delay?

$$10\% * \frac{current\_queue\_length}{max\_queue\_length} \quad (1)$$

We believe the observed user behaviors can help here. Essentially, backfilling favors smaller and shorter jobs in order to efficiently utilize fragmented resources. Our previous observation suggests: when waiting queue grows, users will submit smaller jobs, indicating a higher likelihood of successful backfilling. To better capitalize on this, we can proportionally enable the relaxed backfilling as the queue grows instead of fixing it. For instance, if the original factor is 10%, we will assign such factor as formula (1) to adaptively relax the backfilling. In this way, we maximize the chance of backfilling when it is more favorable, and reduce the chance and avoid the delay when it is less favorable.

Traces	Metrics	Baseline	Adaptive	Improved
Blue Waters	*wait*	7513.02	7520.38	<1%
	bsld	39.02	38.99	<1%
	util	0.7164	0.7165	<1%
	violation	1258.35	1200.81	**5%**
Mira	*wait*	34199.90	36210.37	-6%
	bsld	37.81	39.40	-4%
	util	0.8792	0.8805	<1%
	violation	670.31	344.89	**49%**
Theta	*wait*	51558.62	50732.92	2%
	bsld	96.93	94.61	2%
	util	0.8708	0.8712	<1%
	violation	3911.23	3413.56	**13%**

TABLE II: Job scheduling performance with adaptive relaxing.

We applied the adaptive mechanism to the existing relaxed backfilling mechanism and compared the performance using SchedGym. Since DL workloads do not include *Wall Time* for backfilling, we only include results on Blue Waters, Mira, and Theta in Table II. We can observe the adaptive mechanism is able to effectively reduce the delay of high priority jobs by (*violation*) 5% on Blue Waters, 49% on Mira, and 13% on Theta. More importantly, we show that the adaptive mechanism, although significantly reduces the delay on waiting jobs and violations, it does not impact the effectiveness of relax backfilling itself. The impacts on various scheduling metrics, such as waiting time, bounded job slowdown, and utilization, are relatively small or even rather slightly positive. For example, it improves original relax backfilling less than 1% on Blue Waters and 2% in Theta. In 6% larger job waiting time case on Mira, the adaptive relaxed backfilling reduces the

violations by 49%.

VII. Related Work

Various job characterizations have been done to understand HPC jobs [8], [29], [31], [44], [2], [20], [32], [43]. Patel et al. [29] analyzed a decade of HPC jobs to support several long-standing traditional knowledge and identify many previously unknown trends and their consequences. Di et al. [8] study the impact of the system's events on the jobs' execution to understand failures and enhance the system's reliability. There are also studies on investigating the characteristics of DL workloads in large-scale clusters [18], [17]. Jeon et al. [18] examined DL job traces from a Microsoft cluster to support the influence of DL training job locality on GPU usage and identify the reasons behind job failures. Hu et al. [17] characterized DL jobs from SenseTime for better designs of efficient GPU schedulers. In contrast to previous research, our study performs a comparative analysis across different system types, elucidating the fundamental similarities and disparities between HPC and DL workloads. We offer a thorough examination of job geometries, failure statuses, and user behavior patterns within the context of diverse systems. Moreover, we leverage these shared traits to enhance job runtime prediction performance and bolster job scheduling efficiency. Our findings demonstrate that these insights can inform future resource management strategies for HPC clusters, particularly in light of the escalating prominence of DL workloads.

VIII. Conclusion

In this paper, we perform a cross-system analysis of job characterization for five representative real-world clusters, covering a classic HPC, classic Deep Learning, and hybrid cluster setups, aiming to better understand how the emerging DL workloads would impact HPC systems scheduling. Our eight takeaways highlight interesting differences and similarities among these systems, guiding us to design more efficient job schedulers for the future HPC systems. Based on these observations, we further introduce two use case studies (*job run time prediction* and *adaptive relaxed backfilling*) to effectively improve existing job scheduling. All our data processing logic and simulator are publicly available at [omitted]. We expect they can benefit researchers in designing more efficient HPC schedulers for the future.

Acknowledgments

We sincerely thank the anonymous reviewers for their valuable feedback. This work was supported in part by National Science Foundation (NSF) under grants CNS-2008265 and U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR), under contract DE-AC02-06CH11357. We acknowledge Argonne Leadership Computing Facility for providing the Mira System Log.

References

[1] William Allcock, Paul Rich, Yuping Fan, and Zhiling Lan. Experience and practice of batch scheduling on leadership supercomputers at argonne. In *Job Scheduling Strategies for Parallel Processing: 21st International Workshop, JSSPP 2017, Orlando, FL, USA, June 2, 2017, Revised Selected Papers 21*, pages 1–24. Springer, 2018.

[2] George Amvrosiadis, Jun Woo Park, Gregory R Ganger, Garth A Gibson, Elisabeth Baseman, and Nathan DeBardeleben. On the diversity of cluster workloads and its impact on research results. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 533–546, 2018.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[5] Jim Collins. Passing the torch from intrepid to mira. https://www.alcf.anl.gov/news/passing-torch-intrepid-mira, 2023.

[6] HPC community code. https://bluewaters.ncsa.illinois.edu/community-codes, 2023.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[8] Sheng Di, Hanqi Guo, Eric Pershey, Marc Snir, and Franck Cappello. Characterizing and understanding hpc job failures over the 2k-day life of ibm bluegene/q system. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 473–484. IEEE, 2019.

[9] National Ignition Facility. National ignition facility achieves fusion ignition. https://www.llnl.gov/news/national-ignition-facility-achieves-fusion-ignition, 2022.

[10] Yuping Fan, Zhiling Lan, Paul Rich, William E Allcock, Michael E Papka, Brian Austin, and David Paul. Scheduling beyond cpus for hpc. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, pages 97–108, 2019.

[11] Yuping Fan, Paul Rich, William E Allcock, Michael E Papka, and Zhiling Lan. Trade-off between prediction accuracy and underestimation rate in job runtime estimates. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 530–540. IEEE, 2017.

[12] Dror G Feitelson. Metrics for parallel job scheduling and their convergence. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 188–205. Springer, 2001.

[13] Diandian Gu, Yihao Zhao, Yinmin Zhong, Yifan Xiong, Zhenhua Han, Peng Cheng, Fan Yang, Gang Huang, Xin Jin, and Xuanzhe Liu. Elasticflow: An elastic serverless training platform for distributed deep learning. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 266–280, 2023.

[14] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Harry Liu, and Chuanxiong Guo. Tiresias: A gpu cluster manager for distributed deep learning. In *NSDI*, volume 19, pages 485–500, 2019.

[15] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[16] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[17] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[18] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of {Large-Scale}{Multi-Tenant}{GPU} clusters for {DNN} training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 947–960, 2019.

[19] Weile Jia, Han Wang, Mohan Chen, Denghui Lu, Lin Lin, Roberto Car, E Weinan, and Linfeng Zhang. Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning.

In *SC20: International conference for high performance computing, networking, storage and analysis*, pages 1–14. IEEE, 2020.

[20] Wayne Joubert and Shi-Quan Su. An analysis of computational workloads for the ornl jaguar system. In *Proceedings of the 26th ACM international conference on Supercomputing*, pages 247–256, 2012.

[21] Elliot Kolker-Hicks, Di Zhang, and Dong Dai. A reinforcement learning based backfilling strategy for hpc batch jobs. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pages 1316–1323, 2023.

[22] Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, et al. Exascale deep learning for climate analytics. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 649–660. IEEE, 2018.

[23] Baolin Li, Rohin Arora, Siddharth Samsi, Tirthak Patel, William Arcand, David Bestor, Chansup Byun, Rohan Basu Roy, Bill Bergeron, John Holodnak, et al. Ai-enabling workloads on large-scale gpu-accelerated system: Characterization, opportunities, and implications. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1224–1237. IEEE, 2022.

[24] David A Lifka. The anl/ibm sp scheduling system. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 295–303. Springer, 1995.

[25] Uri Lublin and Dror G Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, 2003.

[26] Ahuva W. Mu'alem and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE transactions on parallel and distributed systems*, 12(6):529–543, 2001.

[27] NCSA. Blue waters data sets. https://bluewaters.ncsa.illinois.edu/data-sets, 2023.

[28] Adam J Oliner, Ramendra K Sahoo, José E Moreira, Manish Gupta, and Anand Sivasubramaniam. Fault-aware job scheduling for bluegene/l systems. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, page 64. IEEE, 2004.

[29] Tirthak Patel, Zhengchun Liu, Raj Kettimuthu, Paul Rich, William Allcock, and Devesh Tiwari. Job characteristics on large-scale systems: long-term analysis, quantification, and implications. In *SC20: International conference for high performance computing, networking, storage and analysis*, pages 1–17. IEEE, 2020.

[30] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–14, 2018.

[31] Gonzalo P Rodrigo, P-O Östberg, Erik Elmroth, Katie Antypas, Richard Gerber, and Lavanya Ramakrishnan. Towards understanding hpc users and systems: a nersc case study. *Journal of Parallel and Distributed Computing*, 111:206–221, 2018.

[32] Gonzalo Pedro Rodrigo Álvarez, Per-Olov Östberg, Erik Elmroth, Katie Antypas, Richard Gerber, and Lavanya Ramakrishnan. Hpc system lifetime story: Workload characterization and evolutionary analyses on nersc systems. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 57–60, 2015.

[33] Li Ruan, Xiangrong Xu, Limin Xiao, Feng Yuan, Yin Li, and Dong Dai. A comparative study of large-scale cluster workload traces via multiview analysis. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 397–404. IEEE, 2019.

[34] Siddharth Samsi, Matthew L Weiss, David Bestor, Baolin Li, Michael Jones, Albert Reuther, Daniel Edelman, William Arcand, Chansup Byun, John Holodnack, et al. The mit supercloud dataset. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8. IEEE, 2021.

[35] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[36] Mark S Squillante, David D Yao, and Li Zhang. The impact of job arrival patterns on parallel scheduling. *ACM SIGMETRICS Performance Evaluation Review*, 26(4):52–59, 1999.

[37] Wei Tang, Zhiling Lan, Narayan Desai, and Daniel Buettner. Fault-aware, utility-based job scheduling on blue, gene/p systems. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE, 2009.

[38] Theta. Theta. https://reports.alcf.anl.gov/data/theta.html, 2023.

[39] ThetaGPU. Thetagpu. https://reports.alcf.anl.gov/data/thetagpu.html, 2023.

[40] TOP500. Top500. https://www.top500.org/, 2023.

[41] Dan Tsafrir, Yoav Etsion, and Dror G Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):789–803, 2007.

[42] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, pages 1–16, 2013.

[43] Laurens Versluis, Roland Mathá, Sacheendra Talluri, Tim Hegeman, Radu Prodan, Ewa Deelman, and Alexandru Iosup. The workflow trace archive: Open-access data from public and private computing infrastructures. *IEEE Transactions on Parallel and Distributed Systems*, 31(9):2170–2184, 2020.

[44] Feiyi Wang, Sarp Oral, Satyabrata Sen, and Neena Imam. Learning from five-year resource-utilization data of titan system. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–6. IEEE, 2019.

[45] William A Ward Jr, Carrie L Mahood, and John E West. Scheduling jobs on parallel systems using a relaxed backfill strategy. In *Job Scheduling Strategies for Parallel Processing: 8th International Workshop, JSSPP 2002 Edinburgh, Scotland, UK, July 24, 2002 Revised Papers*, pages 88–102. Springer, 2002.

[46] Qizhen Weng, Lingyun Yang, Yinghao Yu, Wei Wang, Xiaochuan Tang, Guodong Yang, and Liping Zhang. Beware of fragmentation: Scheduling {GPU-Sharing} workloads with fragmentation gradient descent. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 995–1008, 2023.

[47] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 595–610, 2018.

[48] Xu Yang, Zhou Zhou, Sean Wallace, Zhiling Lan, Wei Tang, Susan Coghlan, and Michael E Papka. Integrating dynamic pricing of electricity into energy aware scheduling for hpc systems. In *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2013.

[49] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing: 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003. Revised Paper 9*, pages 44–60. Springer, 2003.

[50] Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. Rlscheduler: an automated hpc batch job scheduler using reinforcement learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.

[51] Di Zhang, Dong Dai, and Bing Xie. Schedinspector: A batch job scheduling inspector using reinforcement learning. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, pages 97–109, 2022.

[52] Yanyong Zhang, Mark S Squillante, Anand Sivasubramaniam, and Ramendra K Sahoo. Performance implications of failures in large-scale cluster scheduling. In *Job Scheduling Strategies for Parallel Processing: 10th International Workshop, JSSPP 2004, New York, NY, USA, June 13, 2004. Revised Selected Papers 10*, pages 233–252. Springer, 2005.