PAL: A Variability-Aware Policy for Scheduling ML Workloads in GPU Clusters

Rutwik Jain, Brandon Tran, Keting Chen, Matthew D. Sinclair and Shivaram Venkataraman
Computer Sciences Department, University of Wisconsin-Madison
Madison, United States of America
Email: {rnjain, bqtran2, kchen346}@wisc.edu, {sinclair, shivaram}@cs.wisc.edu

***Abstract*—Large-scale computing systems are increasingly using accelerators such as GPUs to enable peta- and exa-scale levels of compute to meet the needs of Machine Learning (ML) and scientific computing applications. Given the widespread and growing use of ML, including in some scientific applications, optimizing these clusters for ML workloads is particularly important. However, recent work has demonstrated that accelerators in these clusters can suffer from performance variability and this variability can lead to resource under-utilization and load imbalance. In this work we focus on how clusters schedulers, which are used to share accelerator-rich clusters across many concurrent ML jobs, can embrace performance variability to mitigate its effects. Our key insight to address this challenge is to characterize which applications are more likely to suffer from performance variability and take that into account while placing jobs on the cluster. We design a novel cluster scheduler, PAL, which uses performance variability measurements and application-specific profiles to improve job performance and resource utilization. PAL also balances performance variability with locality to ensure jobs are spread across as few nodes as possible. Overall, PAL significantly improves GPU-rich cluster scheduling: across traces for six ML workload applications spanning image, language, and vision models with a variety of variability profiles, PAL improves geomean job completion time by 42%, cluster utilization by 28%, and makespan by 47% over existing state-of-the-art schedulers.**

***Index Terms*—GPGPU; Cluster Scheduling; Machine Learning; Performance Variability; Power Management**

I. Introduction

Artificial intelligence (AI) and machine learning (ML) have transformed society with significant improvements for a wide range of tasks [1]. This tremendous transformative effect has been enabled by a virtuous synergy of (1) better hardware systems, (2) larger datasets, and (3) improved ML models (e.g., Transformers) and algorithms that further benefit from more efficient hardware and larger datasets. ML is also increasingly impacting scientific applications [2]–[4]: ML models are either replacing or supplementing traditional computing methods in application domains like molecular dynamics (e.g., DeePMD [5], [6]), protein folding (e.g., OpenFold2 [7]), and scientific AI models (e.g., AuroraGPT [8]).

However, meeting the computing needs of ML applications introduces new challenges. With the slowing of Moore's Law and end of Dennard's Scaling, large-scale systems are increasingly turning towards heterogeneous accelerators to scale performance, especially for ML workloads. For example, large computing centers including cloud providers [9]–[11] have deployed large accelerator-rich clusters that provide peta- or exa-scale levels of compute. These systems often contain hundreds to tens of thousands of accelerators and are usually shared between many users. Thus, *cluster schedulers* need to handle large, accelerator-heavy ML workloads, while also aiming to reduce the time-to-solution of individual jobs and maintaining high resource utilization.

However, achieving high resource utilization for ML workloads is challenging in the face of performance variability of accelerators. Prior studies [12]–[19] have found that large clusters with accelerators like general-purpose GPUs (GPGPUs) exhibit significant *performance variability*, both within and across machines (discussed further in Section II-A). Prior work found that one of the main variability sources is power management (PM) in accelerators, which can lead to power and frequency variations across nodes [18]. Performance variability also causes resource under-utilization for multi-GPU jobs since all of them must wait for the slowest one to complete due to the bulk synchronous programming (BSP) model used in data-parallel ML workloads [20]. Consequently, performance variability makes it challenging for ML workloads to achieve repeatable, high performance.

To overcome this challenge, our goal is to **harness and embrace performance variability**. Specifically, we propose to redesign scheduling policies for GPU clusters to consider performance variability. Our key insight for designing a better policy comes from the fact that performance variability is *application-specific*. For example, prior work found that compute-intensive workloads such as training a ResNet-50 ML model had significant variability (22% geomean variability, max 3.5×). Conversely, memory-intensive workloads such as PageRank had very low variability (1%) [12], [18]. Thus, we can create new policies that take into account both the level of performance variability in a given cluster and how different applications are impacted by performance variability in that cluster.

We achieve our goal by (1) characterizing hardware performance variability by running a wide variety of single- and multi-GPU ML applications on the Texas Advanced Computing Center's (TACC) [21] Longhorn and Frontera clusters, (2) building application-specific performance variability profiles, and (3) designing new placement policies that utilize (1) and (2). Specifically, we propose a new job placement policy, **PM-First**, which considers PM-induced variability as the

primary factor when assigning GPUs to jobs. Our policy uses application profiles to give preferred GPUs to applications that are most sensitive to variability. To ensure our policy can scale to handle large GPU clusters, we identify which GPUs exhibit similar performance variability and use K-Means clustering to group such GPUs together.

While PM-First exclusively focuses on performance variability to make allocation decisions we find this sometimes results in sub-optimal schedules. For example, if the accelerators with similar performance variability profiles are widely distributed across the cluster, scheduling to minimize or reduce performance variability may lead to significant overhead from inter-node communication (e.g., when weights are updated at the end of ML training epochs). To address this challenge, we extend PM-First's algorithm to consider *both* performance variability and locality when making its scheduling decisions. We call this second approach **PAL (Performance Variability & Locality)** since it aims to balance both concerns when scheduling multi-GPU jobs in the cluster. PAL co-optimizes for both locality and variability by estimating the combined effect of both for every possible GPU allocation for a given job. However, selecting the best allocation considering both factors can be expensive for a large cluster. Thus, we propose an efficient mechanism where we construct and traverse a locality-variability matrix (L$\times$V matrix). The L$\times$V-matrix is succinct and its size is bound by the number of locality levels in the cluster (i.e., the network topology hierarchy) and the number of K-Means clusters used for grouping PM scores. We show that this allows PAL to behave the same as PM-First for variability-sensitive jobs, and make locality-first allocations for jobs that need to prioritize network communication.

Given the importance of ML workloads running in large-scale clusters, prior work has also examined scheduling policies for ML workloads [22]–[25]. However, almost all of these scheduler designs are agnostic to GPU variability and assume that iso-architecture GPUs deliver equal performance. One notable exception is Gavel [26], which considers performance heterogeneity but only across different accelerator architectures in heterogeneous clusters. Thus, to the best of our knowledge, our work is the first to make cluster schedulers aware of iso-architecture GPU performance variability. We discuss this and other related work further in Section VI.

To evaluate the efficacy of PM-First and PAL, we integrated them into Blox [27], a state-of-the-art toolkit that supports many modern scheduling and placement schemes for ML workloads. While our policies can potentially also benefit other types of workloads such as scientific computing, their evaluation is left for future work. We compare our PM-First and PAL placement policies with widely used locality-based placement policies from Tiresias [23] and Gandiva [28], which perform job packing. We also evaluate our placement with different scheduling policies, including FIFO, LAS, and SRTF. Overall, PM-First and PAL significantly improve cluster scheduling for a variety of ML workload traces from prior work [29], [30]. For example, PM-First improves geomean 99th percentile job completion time (JCT) by 40%, average JCT by 40%, utilization by 26%, and makespan by 44% over Tiresias. Moreover, by balancing both performance variability and locality, PAL further improves on PM-First. Compared to Tiresias, PAL improves geomean 99th percentile JCT by 41%, average JCT by 42%, and makespan by 47%. PM-First and PAL benefits are especially large when workloads have a large proportion of multi-GPU jobs – the increasingly common case as ML model sizes continue growing. Thus PAL's benefits are likely to scale further with future workloads. Finally, we also validate our approach using a 64-GPU TACC Frontera cluster and find that PAL outperforms Tiresias by 24% in terms of average JCT.

II. Background

A. GPU Performance Variability

GPUs in large clusters such as datacenters and supercomputers exhibit *variability* in performance, despite having the same underlying architecture and being similarly configured. Prior studies [12]–[14], [16]–[18] have examined and characterized performance, power, and temperature variability among GPUs at scale. For example, Sinha, et al. profiled 5 large, GPU-rich clusters with $400-27000$ identical GPUs and identified significant performance variability, both within and across machines. Similar to prior work [12], they found that different applications exhibit different extents of performance variability at scale. In particular, compute-bound workloads like ResNet-50 see much more variation (22% geomean, up to 3.5$\times$) than memory-intensive workloads like PageRank (1% geomean).

Performance variability occurs for a number of reasons in these systems. Static effects at the hardware level such as process variation and die binning cause inherent manufacturing variability among GPUs, while power and temperature limits and associated PM algorithms cause dynamic variation. In HPC systems, non-uniformity in cooling across nodes can also cause thermal throttling, increasing performance variability [31]. This variability is not transient either: performance variability is consistent over time, and ill-performing GPUs are consistently ill-performing [18]. Moreover, not all the performance variability can be explained by temperature differences or cooling sources, and the variability is consistent across different days of the week, times of day, and GPU vendors. Moreover, as discussed in Section I massively parallel workloads that synchronously utilize multiple GPUs are bottlenecked by the worst-performing GPU – reducing cluster underutilization and load imbalance, hurting overall throughput and efficiency of these large-scale systems. Consequently, this application-specific performance variability affects application performance across runs on the same cluster [13], [18] and is a growing problem for accelerator-rich systems.

B. Cluster Scheduling

ML workloads are increasingly being run on shared, GPU-rich clusters. Thus, efficient scheduling is necessary to minimize ML training time and efficiently utilize cluster resources. Collectively, the cluster scheduler must decide which job(s) to schedule at a given time and what resources they should be

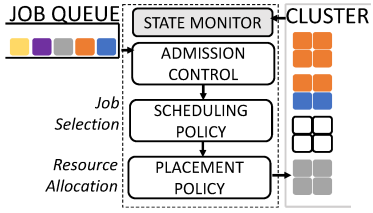


Fig. 1: Modular view of Blox job scheduling.

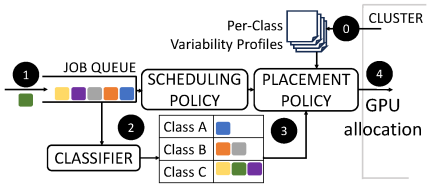


Fig. 2: Variability-aware scheduling overview.

given, respectively. The architecture of most cluster schedulers can be broken up into modules that determine job admission, job selection, and resource allocation [27].

Figure 1 provides a broad overview of these modules. All incoming jobs are put into a queue and admitted based on an admission control policy. Schedulers typically admit jobs that do not adversely impact the performance of currently running jobs and do not violate resource constraints [23], [29]. The scheduling policy receives these accepted, active jobs to schedule in each epoch or *scheduling round*. The active jobs are then assigned priorities and reordered appropriately by the *scheduling policy*, as per the scheduling objective. Finally, this ordered job queue is forwarded to the *placement policy*, which determines what resources (e.g., GPU(s)) should be allocated to a job based on the state of the cluster that the scheduler actively monitors. While the scheduling policy selects which jobs to run at a given epoch, the placement policy determines which GPUs to run them on. Job selection is largely orthogonal to our work, since we are focused on incorporating GPU performance variability into allocation decisions. Thus we focus on the scheduler's placement policy. As discussed further in Section VI, current state-of-the-art placement policies are agnostic to GPU performance variability. When allocating GPU resources to jobs, they assume that iso-architecture GPUs deliver equal performance. Therefore, GPU placement policy designs that harness GPU performance variability information when making allocation decisions are needed.

III. Design

We propose placement algorithms that make GPU variability a first-class citizen when determining GPU job allocations. We leverage a key insight from previous work [12], [18]: *GPU variability is application-specific; e.g., compute-bound workloads exhibit higher variability than memory-bound ones.* This implies that memory-bound jobs can use GPUs that widely vary for compute-intensive jobs without significant performance loss, and allow compute-intensive jobs to utilize GPUs with similar variability.

Figure 2 provides an overview of our approach. We perform offline profiling to gather variability profiles from the target

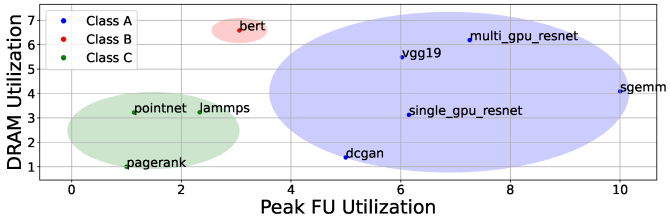


Fig. 3: Classification of applications using 2-dimensional clustering over the $\text{Util}_{\text{DRAM}} \times \max(\text{Util}_{\text{FU}})$ space.

cluster and pass this information to our placement policy ⓪. Since performance variability is application-specific, we create and use an application classifier (Section III-A) that groups similarly behaving applications into a small number of classes, based on compute intensity ②. Figure 2 shows various jobs grouped into three classes, A, B, and C, with A representing the most compute-intensive, and C representing the most memory-bound applications. Section III-A further discusses the classifier. Arriving jobs are put into the job queue ① and the classifier tags these applications with a suitable class ②. The scheduler receives active jobs and assigns them priorities based on its scheduling objective. This sorted queue of jobs is forwarded to the placement policy, which determines what resources should be allocated to a job. Our placement policy receives the job's performance class ③, allowing it to make application-specific decisions; and profiled variability data ⓪ which it uses to make variability-aware allocations ④. We propose two algorithms for placement: PM-First (Section III-B) and PAL (Section III-C).

A. Classification Layer

GPU clusters concurrently run multiple jobs, and new ones arrive frequently. Moreover, new applications, particularly ML models, are emerging frequently and often change the footprint of workloads run on the cluster [10], [32]–[34]. It is infeasible to profile such a large range of applications, especially at scale, to measure performance variability across thousands of GPUs. To reduce the number of required application profiles, we use a classifier that groups similarly behaving applications into a small number of classes. This significantly reduces the amount of profiling data we need to collect.

To classify the variability of the applications we study (Section IV-B) we leverage prior work's application classification scheme [35], [36]. Like Guerreiro, et al. [36], we use `nsight compute` [37] to measure workloads' DRAM utilization ($\text{Util}_{\text{DRAM}}$) and Peak Functional Unit utilization ($\max(\text{Util}_{\text{FU}})$). Each workload corresponds to a point in the 2-dimensional $\text{Util}_{\text{DRAM}} \times \max(\text{Util}_{\text{FU}})$ space. Figure 3 shows the applications we consider in this 2D space. Then we perform K-Means clustering to obtain ordered classes. K can be appropriately set to choose the number of classes for the classifier. For example, by setting $K = 3$ in Figure 2 jobs are assigned one of three classes: A, B, or C, where A is the most sensitive to variability and C is the least sensitive to variability. For a new application or an application with different input parameters/datasets, we profile the application

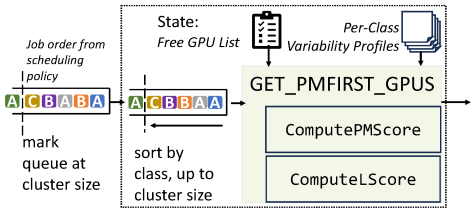


Fig. 4: PM-First Placement Policy with job queue reordering based on placement priority.

and assign it to the cluster it is closest to in the 2D space. We retain this three-class example to illustrate the design of our policies in subsequent sections.

B. PM-First Placement Policy

PM-First gives PM-induced variability first-order precedence when assigning GPUs to jobs. To carry out PM-First allocations, we associate a PM-Score with each GPU, which indicates how slow or fast the GPU is relative to the median GPU in the cluster. The PM-Scores for a GPU are computed for each job class since each class has a different variability profile. We perform application-specific variability profiling by running an application on each GPU to collect performance metrics and use these profiled performance values normalized to the median GPU on the cluster. For example, a PM-Score of 1.5 for a GPU g means that a job's iteration time will be slowed down by 50% running on g compared to the median performing GPU. Since the PM-Score values for a GPU correspond to normalized execution time on the GPU, we refer to GPUs with lower PM-Scores as well-performing GPUs.

Since class A applications are most sensitive to variability, we assign class A jobs the highest placement priority, then class B, and so on. PM-First placement follows a greedy algorithm where the GPUs with the lowest PM-Scores are assigned to jobs with the highest placement priority. This placement priority is different from the scheduling priority produced by the scheduling policy. Figure 4 shows an example of a 6-job queue sorted by the scheduling policy and sent to our placement policy for GPU assignment. The class A jobs in the queue have higher placement priority since we want to provide a larger set of well-performing GPUs to these jobs. However, we must respect the scheduling priorities set by the scheduling policy. To ensure this we mark the job queue when the sum of GPU demands of jobs in the queue exceeds the cluster size. In Figure 4, the GPU demand exceeds cluster size after the first 5 jobs. Thus, the scheduling policy must guarantee that these 5 jobs be scheduled in this round. Therefore, we only sort this truncated queue by job class, allowing compute-intensive class A jobs to select GPUs first, then class B, and so on. This prevents an incoming class A job (marked green in Figure 4) from getting dispatched out of turn. By separating scheduling priority and placement priority, we honor the scheduling policy's guarantee of which jobs to service in a given round, while still allowing GPU allocations to be done using a class-based priority.

Algorithm 1: `GET_PMFIRST_GPUS`
PMFirst Selection Algorithm

Input: Free GPU List G_{free}
Job Class C_j
Job GPU Demand N_j
Variability Profile $V_{profile}$
Output: GPU Allocation $Alloc$

Function `GET_PMFIRST_GPUS` (G_{free}, C_j,N_j,$V_{profile}$):
  `// Get per-GPU PM-Scores` V_i `corresponding to job class`
  foreach *gpu* $i \in G_{free}$ **do**
    $V_i \leftarrow$ `ComputePMscore` ($V_{profile}$, C_j);
  end
  `// Sort free GPU list by PM-Score, from best to worst`
  $G_{free} \leftarrow$ Sort (G_{free}, V_i, descending);
  `// Select the top` N_j `number of GPUs`
  $Alloc \leftarrow G_{free}[:N_j]$;
  `// Remove from free GPU list`
  `MarkGPUsInUse` ();
  return *allocation*;

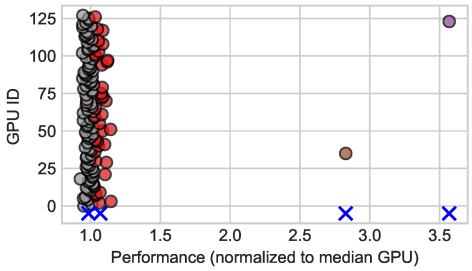


Fig. 5: Example showing clustering on a 128-GPU cluster for a class A application. A blue cross marks each bin's centroid.

Algorithm 1 shows how PM-First allocates resources for a given job j with a GPU demand N_j. We compute the PM-Scores V_i for each GPU i using the variability profile for job j's class (`ComputePMScore`). Then we sort the free list of available GPUs by their PM-Scores, from best to worst. The policy picks the first N_j GPUs to satisfy the GPU demand for job j. This process continues for the next job(s) in the modified job schedule.

Using fine-grained variability information when scheduling a large-scale system with thousands of GPUs could be expensive. For instance, Oak Ridge National Lab's (ORNL's) Summit supercomputer has over 27000 GPUs, and assigning a PM-Score to each of these GPUs and tracking them during run time adds significant overhead to the scheduler. Thus, we use K-Means clustering for each class to bin GPU variability values into a set of PM-Scores.

Figure 5 shows our clustering method employed on the variability profile of a compute-intensive application, ResNet-50, for a 128-GPU cluster. The x-axis shows average iteration time normalized to the median GPU of the cluster. Most GPUs belong to the first 2 clusters close to the median, while some outliers are more than 2.5× slower than the median. We associate the PM-score for all GPUs in a given bin with the PM-score of the centroid. For example, in Figure 5 all GPUs of the gray cluster get assigned a PM-Score of 0.99.

If we pick very small values for K (the number of bins produced by clustering), we lose fine-grained variability information, and PM-First cannot differentiate between GPUs that deliver different performances. Conversely, very high K values overestimate the impact of variability, making PM-First more selective in picking GPUs than necessary. Thus, we need to determine optimal K values for each class. We select the optimal K-value using the standard silhouette score method [38]. However, since the variability data has some extreme outliers, particularly for compute-intensive applications, this adversely impacts silhouette coefficients. We separate $> 3\sigma$ outliers when computing silhouette scores and sweeping through K from 2 to 11. We select the K value that gives silhouette scores as close to +1 as possible for all bins so that we get distinct and relatively well-separated bins. We similarly determine an optimal K value for the set of outliers. Note the placement policy does not ignore $> 3\sigma$ outliers; they are only removed for the silhouette score analysis. These extreme outliers are assigned their own PM-score equal to the GPU's normalized performance.

C. PAL Placement Policy

While the PM-First policy factors variability into its decision-making, it ignores communication overheads that may occur due to ineffective packing. Thus, PM-First works well for applications that are sensitive to variability (e.g., class A), but not for those that are less impacted by variability (typically class C). Accordingly, our PAL placement policy co-optimizes for locality and variability by observing their combined effects, ensuring that PAL prioritizes either packing or variability depending on what is more important.

1) Combined Slowdown and $L \times V$ Matrix: Large-scale systems typically have a flat network topology without much oversubscription. For example, TACC Frontera uses a Mellanox interconnect in a fat tree topology with an oversubscription of 22/18 [21]. For such systems, where there is no complex routing or multiple hops between nodes of the same layer, we use a simplified locality model. A multi-GPU job incurs a performance penalty L_{across} if its allocation spills across nodes and suffers no performance degradation if the allocation is within a node ($L_{within} = 1.0$). If a job is running with a set of GPUs G and the GPUs are spread across more than one node, then the job's modified iteration time is:

$$t_{iter} = L_{across} \times \max_{g \in G}(V_g) \times t_{iter}^{orig} \quad (1)$$

where t_{iter}^{orig} is the job's original iteration time, as specified in Section II, V_g is the variability or PM-Score of the gth GPU and L_{across} is the inter-node locality penalty. Section IV provides more details on estimating a cluster's locality penalty.

To get better performance for jobs, we need to minimize the combined slowdown due to both variability and locality penalties. We denote this as the LV-Product:

$$\min \text{LV-Product} = \min(L_1 \times \max_{g \in G} V_g)$$

To minimize this product, we construct an $L \times V$ matrix for each job class at design time based on the profiled variability and the locality penalty. For example, consider the following $L \times V$ matrix with 4 bins for PM-Scores $V_1 = 0.89, V_2 = 0.94, V_3 = 1.06$, and $V_4 = 2.55$, and a constant inter-node locality penalty $L_{across} = 1.5$.

$$\mathbf{L} \times \mathbf{V} = \begin{matrix} \\ L_{\text{within}}(1) \\ L_{\text{across}}(1.5) \end{matrix} \begin{matrix} \begin{matrix} V_1(0.89) & V_2(0.94) & V_3(1.06) & V_4(2.55) \end{matrix} \\ \begin{bmatrix} 0.89 & 0.94 & 1.06 & 2.55 \\ 1.34 & 1.41 & 1.59 & 3.88 \end{bmatrix} \end{matrix}$$

The matrix entries represent the LV-Product we want to minimize. Each entry corresponds to a possible allocation scenario. We "traverse" this matrix from smallest LV-Product to largest, making job allocations to minimize the LV-product (Algorithm 2, line 3). In this example, the $L \times V$ matrix traversal order would be: $(1, 0.89) \rightarrow (1, 0.94) \rightarrow (1, 1.06) \rightarrow (1.5, 1.34) \rightarrow (1.5, 1.41) \rightarrow (1.5, 1.59) \rightarrow (1.5, 3.88)$. In other words, unlike PM-First, PAL allows non-packed allocations only if the first three variability bins cannot provide a packed allocation to service this job. However, PAL prefers a distributed allocation over allocating GPUs from bin 4 which has a very high PM-Score ($V_4 = 2.55$). Moreover, since the $L \times V$ matrix is class specific traversal orders are often unique per-class. This allows PAL to make PM-First decisions for variability-sensitive jobs, and make locality-first allocations for jobs that need to prioritize packing.

Algorithm 2 details the steps to perform this $L \times V$ traversal. For a job requesting N_j-GPUs on a cluster size N, there are ${}^{N}C_{N_j}$ possible GPU allocations. Our inter-node cost model for locality allows us to reduce this search space, since jobs with GPU demand $N_j >$ `NUM_GPUS_PER_NODE` must request multiple nodes and pay the inter-node locality penalty of splitting across nodes. PAL schedules all such jobs using the PM-First policy (Algorithm 2, lines 23-25). A job j requesting N_j GPUs, where $1 < N_j \leq$ `NUM_GPUS_PER_NODE`, traverses the allocation in two ways:

1) (L_{within}, V_i) allocations: PAL needs to prioritize packing while making sure that PM-Score for the allocation is $\leq V_i$. We filter out the free list of GPUs with $V \leq V_i$ and then try to enumerate strictly packed (within-node) allocations within this free list by enumerating all possible packed N_j-sets of GPUs and finding the set with the least variability.
2) (L_{across}, V_i) allocations: we filter out the free list of GPUs with PM-Score $\leq V_i$ and sort them by PM-Score. Since locality cost is acceptable to incur in this state, we pick the first N_j GPUs from this sorted list.

This $L \times V$ matrix traversal only occurs for jobs that require `NUM_GPUS_PER_NODE` or fewer number of GPUs.

IV. Methodology

A. System

We run experiments on both a physical cluster and in simulation. All experiments use Blox [27], an open-source modular toolkit that uses Python and gRPC [39] to support scheduler implementation and testing. The physical cluster

Algorithm 2: PAL_PLACEMENT
PAL Selection Algorithm

Input: Free GPU List G_{free},
Job Class C_j
Job GPU Demand N_j
$L \times V$ matrix
Output: GPU Allocation $Alloc$

Function PAL_PLACEMENT(G_{free}, C_j, N_j, $L \times V$):
  if $1 < N_j \leq$ *NUM_GPUS_PER_NODE* **then**
    for (L_i, V_i) ***in*** traverse($L \times V$) **do**
      if $L_i = L_{within}$ **then**
        // Filter GPUs with PM Scores better or equal to V_i
        $G_{filt} \leftarrow G_{free}[V \leq V_i]$;
        // Enumerate potential packed allocations
        $n \leftarrow$ FindValidNodes (G_{filt}, N_j);
        $PackAlloc \leftarrow$ GenerateCombos(^nC_K);
        // Return one with lowest PM Score
        $Alloc \leftarrow$ GetMinV($PackAlloc$);
        return $Alloc$;
      end
      else if $L_i = L_{across}$ **then**
        // PM-First allocation
        $G_{filt} \leftarrow$ filt (*free_gpu_list,* $V \leq V_i$);
        $Alloc \leftarrow G_{filt}[:N_j]$;
        return $Alloc$;
      end
    end
  end
  else
    // PM-First allocation
    $Alloc \leftarrow$ getPMFirstGPUs();
    return $Alloc$;
  end

TABLE I: List of experiments used in evaluation.

Workload Trace	Cluster Size (NumGPUs)	Experiment	Eval. Section
Sia-Cluster [29]	64	Testbed Evaluation	V-A
Sia-Philly [29], [40]	64	Baseline Simulation Varying Locality Penalty	V-B
Synergy [30]	256	Varying Job Load Varying Schedulers	V-C

experiments are performed on TACC's Frontera supercomputer [21]. Frontera is a mineral-oil cooled GPU subsystem with 360 NVIDIA Quadro RTX 5000 GPUs. Each node has 4 GPUs, with 16GB memory per GPU. We run physical cluster experiments on an 16 node (64 GPU) testbed. We also use larger trace-based simulations to evaluate the behavior of our policies with varying cluster sizes, traces, and scheduling policies. Table I summarizes the key experiments and system details.

1) Baseline Placement Policies: We evaluate PM-First and PAL's performance relative to two baselines: (1) Packed or soft-consolidated placement tries to minimize the number of nodes a job is packed on to reduce communication; and (2) Random or Scattered placement samples a random subset from the free list of GPUs in order to prevent thermal hotspots through unbalanced GPU usage, increase device lifespan, and prioritize performance of CPU-to-GPU communication. However, random placement can sacrifice performance for workloads sensitive to GPU-to-GPU communication.

We further consider two flavors of each of these policies – Sticky and Non-Sticky. In Sticky placement, active jobs cannot be migrated to a different allocation and must continue to run with the same ("sticky") set of GPUs they first get assigned, until the jobs either complete or get preempted through priority lowering. Thus, the Sticky placement policy only re-allocates GPUs to a job once the job moves from suspended to active state. Sticky allocations minimize checkpointing overheads that occur due to migration, but these are typically negligible relative to the overall job run-time. We chose these as our baselines because most state-of-the-art schedulers use one of them. Specifically, we compare against the following configurations:

1) **Tiresias** [23]: performs Packed-Sticky placement
2) **Gandiva** [28]: performs Packed-Non-Sticky placement
3) **Random-Sticky**
4) **Random-Non-Sticky**

Other schedulers also use some variants of these policies. For example, Themis [41] also uses Packed placement, while Amaral, et al. and HotGauge use Random placement [42], [43]. In the remainder of this paper we use **Tiresias** to mean Packed-Sticky placement and **Gandiva** to mean Packed-Non-Sticky placement.

Our **PAL** and **PM-First** placement policies are both Non-Sticky to ensure jobs can migrate to better GPUs in each scheduling round.

2) Scheduling Policies: We evaluate three schedulers PM-First and PAL placement can be attached to:
First-In-First-Out (**FIFO**) scheduler: a well-known greedy approach that prioritizes jobs in order of arrival.
Tiresias/Least Attained Service (**LAS**): implements LAS scheduling with two-level priority queuing [23].
Shortest Remaining Time First (**SRTF**): performs preemptive shortest job first scheduling.

While we present simulation results with all three scheduling policies, we use Tiresias for the TACC Frontera cluster runs. Section V-C compares how the behavior of our placement policy changes when the scheduler is varied.

B. Workloads and Cluster Configuration

1) Simulations: To compare our placement policies against the baselines (Section IV-A1) in simulation we use two sets of workload traces:
Sia-Philly workloads [29] sample jobs from Microsoft's publicly available Philly cluster production traces [40]. Sia derives eight traces of 160 jobs each, submitted over an 8 hour window at a job arrival rate of 20 jobs/hr. We use these traces in the same cluster configuration Sia used: a simulated 16 node system with 4 GPUs per node (64 GPUs). 40% of Sia trace jobs are single-GPU jobs, and the largest multi-GPU jobs request up to 48 GPUs. We report average metrics such as

TABLE II: Models used in real cluster evaluation.

Task	Model	Dataset	Batch Size	Class
Image	PointNet [44]	ShapeNet [45]	32	Class C
Image	vgg19 [46]	ImageNet2012 [47]	32	Class A
Vision	DCGAN [48]	LSUN [49]	128	Class A
Language	BERT [50], [51]	WikiText [52]	64	Class B
Image	ResNet-50 [53]	ImageNet2012 [47]	32	Class A
Language	GPT2 [51]	Wikitext [52]	128	Class B

TABLE III: Applications profiled for PM penalty estimation

Benchmark	Input Size	Clusters Observed
ResNet50 [54]	Train. Set: 1.2M images Batch size: 64	Longhorn Frontera
BERT [55]	Train. Set: 30K words Batch size: 64	Longhorn Frontera
PageRank [56]	643994×643994 659033×659033	Longhorn Frontera

Job Completion Time (JCT) across these 160 jobs, consistent with prior work that used these traces [24], [29].

Synergy workloads preserve the Philly trace's [40] GPU demand and use a Poisson distribution of arrival times to vary job arrival rate. Synergy traces have a higher proportion of single-GPU jobs ($> 80\%$) than Sia-Philly traces. To evaluate Synergy's steady state benefits, like prior work we simulate Synergy on a larger 64 node, 4-GPU per node cluster (256 GPUs) and report average metrics for job IDs 2000 to 3000.

2) Cluster Evaluation on TACC Frontera: Table II lists the jobs we run for our real cluster experiments on the TACC Frontera cluster. As with the Sia simulations, we track average JCT metrics across all 160 jobs.

C. Estimating PM penalty

We perform variability profiling to estimate per-GPU, per-class PM penalties. We use two sets of variability profiles from different systems, one from TACC's Longhorn cluster (NVIDIA V100 GPUs), used for simulations, and one from TACC's Frontera cluster (NVIDIA Quadro RTX 5000 GPUs) used in both simulation and cluster experiments. These profiles are generated by running the same benchmark on all GPUs of the cluster and collecting metrics for kernel duration/iteration runtime. As discussed in Section III-A, we profile one representative application for each class. We used NVIDIA's `nsight compute` [37] to collect performance metrics in milliseconds (ms) for these applications. Since `nsight compute` can only provide 1 sample every ms, we configured our input sizes to ensure that kernel durations were larger than 1ms on the respective GPUs. Table III summarizes the applications we ran and their input configurations on each cluster. Figure 6 and Figure 7 show each application's GPU performance normalized to the application's median observed duration on the cluster. When simulating an N-GPU cluster, we discretely, randomly sample this profiling data without repetition to obtain N PM penalty values for each class and assign them to each GPUs.

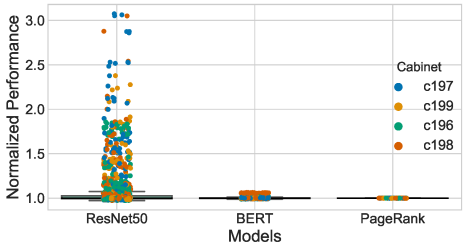


Fig. 6: Normalized Frontera cluster performance variability.

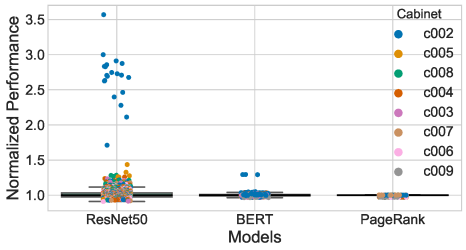


Fig. 7: Normalized Longhorn cluster performance variability.

TABLE IV: Physical cluster & simulation results.

Placement Policy	Avg JCT (hours)		Cluster-to-Sim Diff
	Cluster	Simulation	
Tiresias	1.76	1.56	11%
PAL	1.35	1.16	14%
% Improvement	24%	26%	

For the Frontera testbed implementation, we index into the variability profile using GPU UUID obtained from `nvidia-smi` [57] to get the exact PM penalty value from the variability profiling data. Profiling for a large number of GPUs could be time-consuming, so our variability profiles are static – they are generated at design time and remain constant throughout.

D. Locality Penalty

We estimated the inter-node locality penalty L_{across} by profiling the iteration time for a 4-GPU ResNet-50 [53] job with a batch size of 64 run on all 4 GPUs of a single node versus an 8 GPU ResNet-50 job with a batch size of 128 running on two nodes. The ratio of the average iteration time aggregated from the two profiles gives us an estimate for locality penalty. Using this method we initially estimated a locality penalty to be 1.7 on TACC Frontera. We use this locality penalty in our Synergy simulations, but study different locality penalty values to evaluate how our policies fare in other systems with different costs of distributing jobs across nodes. Our physical experiments showed that inter-node communication costs are not as high on Frontera, and are also model-dependent. Hence, we estimate per-model locality penalties from our physical cluster experiments, and use these in simulation experiments in Sections V-A and V-B.

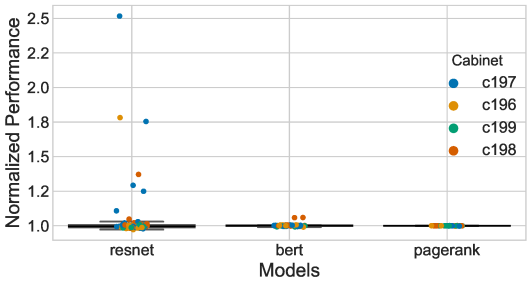


Fig. 8: TACC Frontera 64-GPU testbed variability profiles.

Fig. 9: Cluster and simulation cumulative Job Completion Time distributions.

V. Evaluation

A. Real Cluster Experiments

To ensure our reported gains from Blox (Sections V-B-V-C) are representative and realistic, we first compare **PAL** to **Tiresias**, the best performing baseline, on a physical 64-GPU cluster. Figure 8 shows the variability profile of this 64-GPU test-bed. Since we are running these experiments on a real cluster, we use the exact PM penalties for the 64 GPUs that we used in our experiments.

Figure 9 shows the cumulative distribution of JCTs for the physical cluster experiments as well as this corresponding simulation (Sections V-B-V-C) for the same trace. Broadly, the cluster CDF aligns fairly well with that of simulation for both policies. Overall, PAL reduces average JCT and makespan by 24% and 27%, respectively, over Tiresias' placement policy, whereas simulation predicts a 28% benefit given the variability and locality penalties of the testbed. These results demonstrate the benefit of our approach; even over the best performing baseline PAL's ability to effectively exploit both locality and variability improves performance. Moreover, Table IV summarizes **PAL** and **Tiresias**'s real cluster average JCT results. The difference between the average JCT metric in real and simulated clusters is less than 14% for both policies. This difference can be attributed to per-application estimation of locality penalty in simulation. Overall, given the relatively low error between real and simulated clusters, and Figure 9's close correlation between the JCTs on the cluster and those obtained in simulation, these results demonstrate Blox's fidelity.

Figure 8 shows the observed variability of the 64 GPU subset of Frontera that formed our testbed. Specifically, the 64 GPUs we ran on have 6%, 2.3% and 0.9% variability for

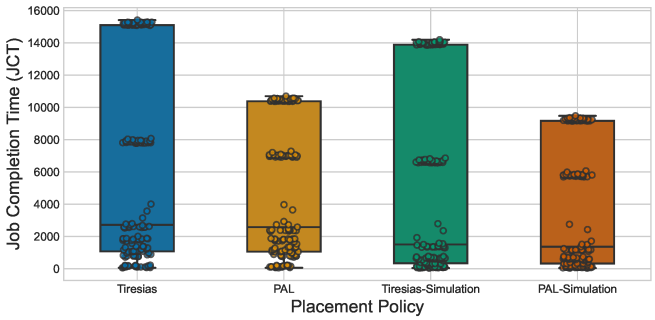


Fig. 10: Physical cluster and simulated **Tiresias** and **PAL** JCT boxplots.

ResNet-50 (class A), BERT (class B), and PageRank (class C) respectively. Even though the testbed exhibits lower variability compared to the overall Frontera profile in Figure 6, our results demonstrate that PAL still significantly improves average JCT over Tiresias. These results both demonstrate the viability of our technique on real GPU clusters and the fidelity of our simulator, which we next utilize to evaluate a wider range of configurations in simulation.

B. Sia-Philly Simulations

Next we evaluate Sia-Philly traces in a simulated 64-GPU cluster with FIFO scheduling using PM penalty profiles from TACC's Longhorn cluster. Further configuration details were described in Section IV-B1, IV-D, and IV-C. Figure 11 compares the average Job Completion Time (JCT), normalized to **Tiresias** or Packed-Sticky placement.

Random & Packed Policies: Since random selection increases the likelihood that the allocated GPUs are ill-performing in terms of variability and/or packing, the **Random** baseline policies often suffer in terms of performance (e.g., workloads 1, 2, 3, 6, and geomean JCT results). Conversely, the packed baselines, namely **Tiresias** and **Gandiva**, improve performance by minimizing the number of nodes a job is packed on, thereby reducing the inter-node locality cost for multi-GPU jobs.

Sticky & Non-Sticky Variants: For most workloads (1, 2, 4, 5, 6, geomean), **Tiresias**' sticky placement outperforms **Gandiva**'s non-sticky GPU selection. This happens because non-sticky placement picks GPUs every scheduling round, increasing the likelihood of jobs picking GPUs with worse PM-Scores for some rounds during their runtime. With Sticky placement jobs that pick GPUs with high PM-Scores get consistent slowdowns throughout their runtime. However, because Sticky placement keeps the same GPUs throughout execution, there are fewer such jobs compared to Non-Sticky placement.

PM-First & PAL: Overall, both **PM-First** and **PAL** both outperform the baseline placement policies: PM-First improves average JCT by 40% geomean (min 5%, max 59%) while PAL improves average JCT by 43% geomean (mix 21%, max 59%) compared to **Tiresias**. **PM-First** and **PAL** also improve makespan by 44% and 47% respectively over **Tiresias**. To better understand **PM-First**'s and **PAL**'s range of benefits, we

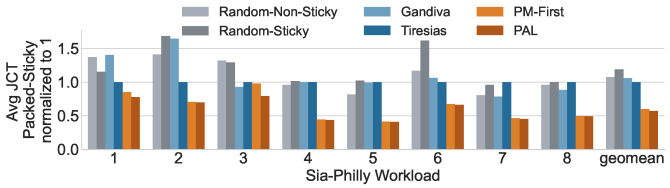


Fig. 11: Average JCT, normalized to **Tiresias** placement, for different Sia-Philly workloads on a 64-GPU cluster with FIFO scheduling policy.

Fig. 12: Comparing wait times under different placement policies for workload traces 3 and 5. For brevity we only compare PM-First, PAL, the best-performing baseline (**Tiresias**).

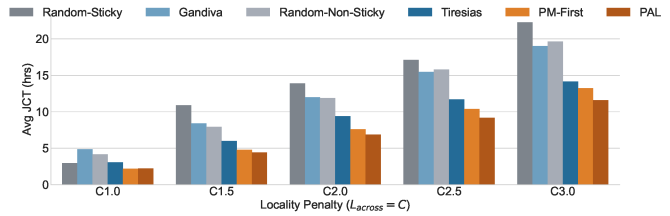


Fig. 13: Average JCT for Sia workloads when inter-node locality penalty varies from 1.0 to 3.0.

examined the GPU demand distribution for workload traces 3 and 5, which provide the best and the worst improvements, respectively, over **Tiresias**. Both workloads have nearly 40% single-GPU jobs, but workload 5 also has some very large multi-GPU jobs – e.g., up to 48 GPU jobs that occupy 75% of the cluster when they are scheduled and increase the waiting times for subsequent, queued jobs (unsurprisingly, workload 5 has longer wait times in Figure 12). Consequently, workload 5's wait times generally increase for subsequent jobs, as expected with a FIFO scheduler. For example, in workload 5 an ImageNet job that requests 48 GPUs arrives early (job ID 19) and blocks subsequent jobs from getting sufficient cluster resources, significantly increasing waiting times. Thus, despite high contention, PAL and PM-First policies more efficiently manage resource allocation and drain the job queue faster than **Tiresias**, reducing waiting time and providing much larger benefits for workload 5. Conversely, in workload 3 long-running jobs with large GPU demands only arrive later on (e.g., job ID 60 in a 160 job trace). As a result, workload 3's jobs have lower wait times, reducing PAL's benefits.

1) Varying Locality Penalty: Figure 13 shows how various inter-node locality penalty values affect our policies for the Sia-Philly workloads. As the locality penalty increases, the cost of allocating GPUs across nodes starts to dominate and the placement policies that prioritize packing (**Tiresias** and **Gandiva**) start winning over the **Random** variants.

As the locality penalty increases, the best-performing baseline (**Tiresias**) improves its average JCT and approaches **PM-First**'s and **PAL**'s performance. For example, **PM-First**'s average JCT improvement over **Tiresias** decreases from 30% to 9% as the locality penalty increases from 1.0 to 3.0. Unsurprisingly, this demonstrates that **PM-First** does well when locality penalty is low and **Tiresias** does well when locality penalty increases. Nevertheless, even with a large locality penalty, **PM-First** still outperforms **Tiresias**, showing the value in harnessing performance variability. However, by prioritizing both locality and performance variability for multi-GPU jobs (Section III), **PAL** outperforms both **PM-First** and **Tiresias**: as locality penalty is increased from 1.0 to 3.0, **PAL**'s benefits over **Tiresias** only decrease from from 30% to 20% geomean.[1]

C. Synergy Trace Simulations

With the Synergy traces we vary the job load or arrival rate (jobs/hour) and measure JCTs to evaluate the performance of our policies under varying levels of cluster contention. Figure 14 compares the average JCT at different job loads with various placement policies on a 256-GPU simulated cluster, a constant locality penalty of 1.7, and variability profiles drawn from TACC's Longhorn cluster (Figure 7). Unsurprisingly, **Tiresias** placement again outperforms the other baselines because packing helps avoid paying the high inter-node locality cost for multi-GPU jobs. Comparatively, **PM-First** sees lower improvements with Synergy versus Sia: since Synergy has more single GPU jobs (Section IV-B1), these jobs do not need to be packed. However, unlike **PM-First**, **PAL** successfully prioritizes packed allocations for multi-GPU jobs, where spilling across nodes is prohibitively expensive, **and** makes PM-First allocations for single GPU jobs that need not be packed. Thus, **PAL** still provides the best overall JCT.

Overall, with FIFO scheduling, **PAL** improves the average JCT from 4% to 9% over **Tiresias** as job load varies from 4 jobs/hr to 12 jobs/hr, before benefits stabilize around 8%. These benefits happen despite Synergy having ≈80% short running, single-GPU jobs. For Synergy's multi-GPU allocations, the effect of variability is more pronounced – since these multi-GPU jobs must periodically synchronize across GPUs, their execution is bound by the slowest GPU in their allocation. Thus, **PAL** improves the average JCT of multi-GPU jobs by 5% to 31% over **Tiresias** as job load varies from 4 to 12 jobs/hour. At job loads $>$ 12 jobs/hour, multi-GPU jobs see 22% average JCT benefits with **PAL**.

Figure 15 shows the number of GPUs in use at every scheduling epoch. For job loads under 8 jobs/hour, there is low contention and the cluster remains under-utilized, as seen in the utilization dip at around 1.5×10^6s. For 10 jobs/hour,

[1]We see similar results when sweeping locality penalty for Synergy simulations, not shown due to space constraints.

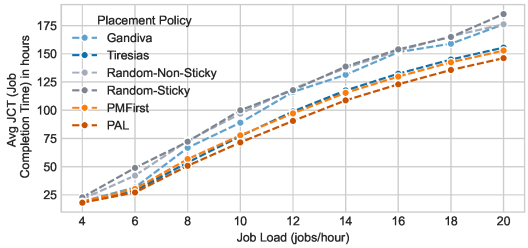
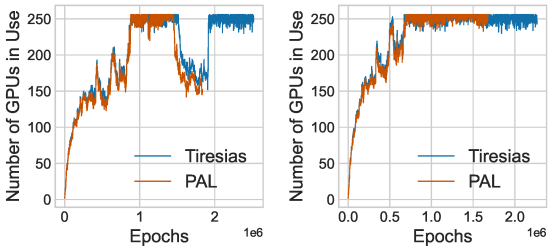


Fig. 14: Average JCT for Synergy traces with FIFO scheduler and varying job load.



(a) 8 jobs/hour (b) 10 jobs/hour

Fig. 15: GPUs in use for different job loads with **Tiresias** and **PAL** placement policies.

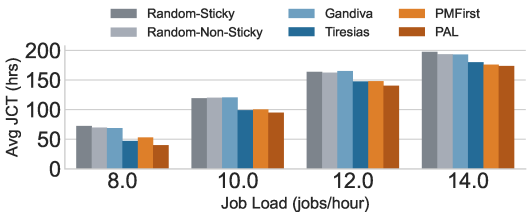


Fig. 16: Average JCT for Synergy traces with LAS scheduler and varying job load.

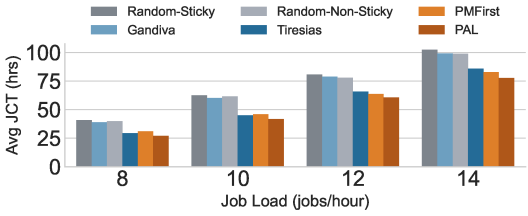


Fig. 17: Average JCT for Synergy traces with SRTF scheduler and varying job load.

the cluster gets saturated early (at around 0.7×10^6s) and all 256 GPUs are consistently busy thereafter. We observe a similar wait time cascading effect as with the Sia traces, where small execution time improvements enable **PM-First** and **PAL** to drain the job queue faster, reducing wait times significantly for subsequent jobs. This can be seen in **PAL**'s utilization pattern, where **PAL**'s utilization "runs ahead" of **Tiresias** (Figure 15) and frees up resources earlier. At low contention levels, queuing time for jobs is low. **PAL** and **PM-First** provide execution time benefits here, but the wait time benefits are limited since most jobs get resources immediately on arrival regardless of placement policy, except for jobs between time 1×10^6 to 1.5×10^6.

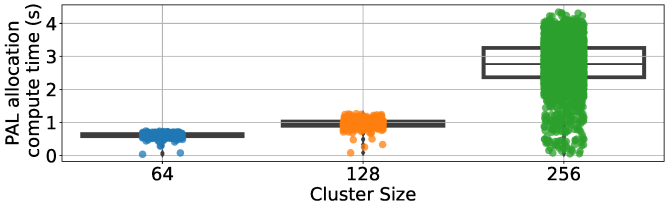


Fig. 18: Placement overheads for PAL policy with varying cluster sizes.

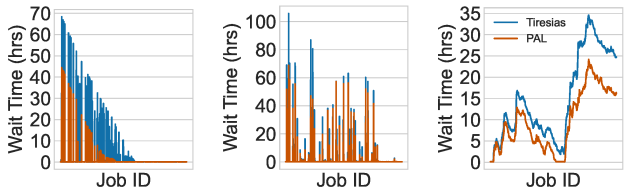
Finally, to estimate **PAL** and **PM-First**'s additional overheads, we measured the time each placement policy takes at every scheduling epoch for different cluster sizes. Figure 18 shows the distribution of policy computation time over all scheduling epochs, for varying cluster sizes. For a 256-GPU cluster, **PAL**'s worst-case scheduling time is 4 seconds (for the very first scheduling epoch) with a median of 2.8 seconds. PM-First's worst-case computation time is 2 seconds, also for the first epoch. Thus, both **PAL** and **PM-First** complete an epoch's GPU assignments within 4 seconds – much smaller than the 300 second epoch duration.

1) Varying Scheduling Policies: SRTF & LAS: Figures 16 and 17 show variation of average JCT as job loads increase from 8 to 14 jobs/hour for LAS and SRTF schedulers respectively. With LAS, the absolute values of JCTs are higher because of higher wait times compared to FIFO. Nevertheless, **PAL** provides up to 15% improvement over **Tiresias**. With SRTF, **PAL** provides up to 10% improvement in average JCT over **Tiresias**. These improvements are higher than FIFO's, and can be attributed to differences in wait time patterns between these scheduling policies.

Figure 19 compares wait times with epochs for the three different schedulers. For LAS, incoming jobs get higher priority than running jobs since they have no attained service – decreasing wait time for successive jobs. The extent of decrease depends on cluster contention levels. Figure 19(a) shows this decrease at 8.0 jobs/hr – wait times go down to 0 for jobs later in the trace. Moreover, the absolute values of wait times are large enough to dominate the overall JCT for these jobs. **PAL** often reduces wait time for these long queued jobs, due to its aforementioned run-ahead effect (Section V-C). Conversely, SRTF has fewer wait time spikes than LAS (Figure 19(b)) but still gets some wait time benefits with **PAL** over **Tiresias**. With FIFO scheduling, wait times progressively increase over time. Thus, its magnitude of wait times is comparatively lower than either SRTF or LAS. As a result, **PAL** correspondingly achieves lower wait time benefits over **Tiresias** with FIFO scheduling.

VI. Related Work

CPU Variability: CPU-based systems have proposed and adopted several solutions to handle variability, but largely focus on different aspects or scales than ours, making them complementary to our proposed approaches. Dynamic load balancing algorithms [58]–[60] use adaptive runtime systems that estimate a task's completion time if it is moved from one



(a) (b) (c)

Fig. 19: Comparing **Tiresias**'s and **PAL**'s wait times for Synergy with (a) LAS, (b) SRTF, and (c) FIFO schedulers.

processor to another for single CPU experiments. Speed-aware solutions identify overloaded nodes as ones with throttled core frequencies [59] while thermal-aware solutions identify hotspots [58], [60]. Conversely, we profile and directly use performance variability for single- and multi-GPU jobs, which requires managing placement and locality across multiple devices for a given job. Other solutions [61] minimize variability within a NUMA node using a runtime-guided search for thread and task socket assignments. However, this does not scale to GPUs with thousands of threads or larger systems with hundreds/thousands of GPUs. Some solutions estimate total consumed power when scheduling jobs to stay within a system-wide power budget. To estimate power variability, jobs either use static profiling tables [62] or power prediction models [61]. While these solutions are power variability aware, neither consider it in their placement policies.

GPU Performance Variability: Prior VLSI and architecture works have developed hardware-level techniques and optimizations to mitigate or tolerate process variation either within a single GPU or components of a single GPU such as compute units, memories, or the register file [17], [63], [64]. However, these solutions are local to the GPU and do not extend to multi-GPU systems. Moreover, they also focus on process variation and do not address dynamic variability due to PM and the overall effect of variability on application performance like our work.

Cluster Schedulers: Unlike batch scheduling mechanisms such as SLURM [65], ML cluster schedulers for GPU-rich systems are tailored towards the unique demands and constraints imposed by ML training, such as checkpointing and periodic job migration. In this work, we specifically focus on such round-based, preemptive schedulers for ML training in large-scale GPU clusters. These schedulers make GPU allocation decisions for competing ML jobs with specific scheduling objectives. Scheduling is an extremely active area of research, with over 23 schedulers proposed for ML training on GPU datacenters since 2022. Thus, there are numerous prior works, as comprehensively summarized by Gao et al. [22]. However, the most relevant ones to our work have developed schedulers to optimize for different objectives, such as JCT, utilization, cost, fairness, and deadlines [23]–[26], [29], [41], [66]. However, unlike PAL, these works are either performance variability agnostic or, like Gavel [26], only consider variability across different GPUs from the same vendor in an HPC system. In comparison, PM-First enables fine-grained iso-architecture variability that allows them to make variability-informed decisions. Moreover, PAL co-optimizes variability with packing to further improve performance.

Placement Policies: Jeon et al. [40] recommended that multi-tenant GPU cluster schedulers should prioritize locality to reduce distributed communication costs for long-running jobs. Accordingly, several schedulers try to pack jobs onto fewer nodes [23], [26], [29], [41]. For example, Tiresias profiles a job to identify its sensitivity to placement and performs packed placement for jobs with high communication overheads [23]. However, all these schedulers are agnostic to GPU variability, which we show in Section V causes PAL to outperform them. Gavel [26] and Sia [29] consider different accelerator architectures on heterogeneous clusters, while Amaral et al. [42] consider heterogeneity in GPU interconnect topology. These solutions also assume that all GPUs of a given architecture deliver equal performance. Thus, to the best of our knowledge, our work is the first to create cluster schedulers aware of performance variability among GPUs in HPC systems with the same architecture and make scheduling decisions informed by this variability.

VII. Conclusion

GPU-rich clusters are becoming increasingly prevalent to meet the computing needs of large-scale ML workloads. However, performance variability can significantly affect their cluster performance, utilization, and load balance – trends that are likely to worsen as ML algorithms continue to scale. Thus, ML schedulers in large scale systems must embrace and harness this variability. Accordingly, we propose **PAL**, which uses application-specific variability characterization to intelligently perform variability-aware GPU allocation. Moreover, PAL co-optimizes job placement for both variability and locality to reduce communication overheads. Overall, across a mix of ML workloads, our evaluation shows that PAL improves on state-of-the-art ML cluster schedulers across a number of metrics. Moreover, we expect HPC and HPC+ML workloads will exhibit similar benefits.

Acknowledgment

We thank the anonymous shepherd and the SC reviewers for their constructive comments and suggestions that improved this work. We express our gratitude to Saurabh Agarwal and Song Bian (University of Wisconsin-Madison) for their support in enabling Blox on Texas Advanced Computing Center's (TACC's) Frontera cluster. We also thank Zhao Zhang (TACC) and Oscar Hernandez (Oak Ridge National Laboratory) for providing us machine access on the clusters we used and helping us run some of the experiments. This work is supported by NSF grant CNS-2312688, and a TACC allocation. Sinclair has an affiliate appointment with AMD Research.

References

[1] N. Benaich and I. Hogarth, "State of AI Report 2022," https://www.stateof.ai/, 2022.

[2] Z. Fan and E. Ma, "Predicting orientation-dependent plastic susceptibility from static structure in amorphous solids via deep learning," *Nature communications*, vol. 12, no. 1, pp. 1–13, 2021.

[3] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[4] J. Kates-Harbeck, A. Svyatkovskiy, and W. Tang, "Predicting Disruptive Instabilities in Controlled Fusion Plasmas through Deep Learning," *Nature*, vol. 568, no. 7753, pp. 526–531, 2019.

[5] H. Wang, L. Zhang, J. Han, and W. E, "DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics," *Computer Physics Communications*, vol. 228, pp. 178–184, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465518300882

[6] J. Zeng, D. Zhang, D. Lu, P. Mo, Z. Li, Y. Chen, M. Rynik, L. Huang, Z. Li, S. Shi, Y. Wang, H. Ye, P. Tuo, J. Yang, Y. Ding, Y. Li, D. Tisi, Q. Zeng, H. Bao, Y. Xia, J. Huang, K. Muraoka, Y. Wang, J. Chang, F. Yuan, S. L. Bore, C. Cai, Y. Lin, B. Wang, J. Xu, J.-X. Zhu, C. Luo, Y. Zhang, R. E. A. Goodall, W. Liang, A. K. Singh, S. Yao, J. Zhang, R. Wentzcovitch, J. Han, J. Liu, W. Jia, D. M. York, W. E, R. Car, L. Zhang, and H. Wang, "DeePMD-kit v2: A software package for deep potential models," *The Journal of Chemical Physics*, vol. 159, no. 5, p. 054801, 08 2023. [Online]. Available: https://doi.org/10.1063/5.0155600

[7] G. Derevyanko, G. Lamoureux, C. Outeiral, T. Oda, F. Fuchs, S. P. Mahajan, J. Moult, J. Haas, P. Maragakis, T. Ruzmetov, and M. AlQuraishi, "OpenFold2: Replicating AlphaFold2 in the Dark," https://lupoglaz.github.io/OpenFold2/, 2023.

[8] R. Stevens, "Argonne's "AuroraGPT" Project," *Trillion Parameter Consortium Seminar*, 2023.

[9] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A Configurable Cloud-scale DNN Processor for Real-time AI," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA. Piscataway, NJ, USA: IEEE Press, 2018, pp. 1–14. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00012

[10] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, "Ten Lessons from Three Generations Shaped Google's TPUv4i," in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ser. ISCA, 2021.

[11] top500.org, "TOP500 List," 2023, https://www.top500.org/lists/top500/list/2023/11/.

[12] J. Coplin and M. Burtscher, "Energy, Power, and Performance Characterization of GPGPU Benchmark Programs," in *IEEE International Parallel and Distributed Processing Symposium Workshops*, ser. IPDPSW, May 2016, pp. 1190–1199. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IPDPSW.2016.164

[13] N. DeBardeleben, S. Blanchard, L. Monroe, P. Romero, D. Grunau, C. Idler, and C. Wright, "GPU Behavior on a Large HPC Cluster," in *Euro-Par 2013: Parallel Processing Workshops - BigDataCloud, DIHC, FedICI, HeteroPar, HiBB, LSDVE, MHPC, OMHI, PADABS, PROPER, Resilience, ROME, and UCHPC 2013, Aachen, Germany, August 26-27, 2013. Revised Selected Papers*, ser. Lecture Notes in Computer Science, D. an Mey, M. Alexander, P. Bientinesi, M. Cannataro, C. Clauss, A. Costan, G. Kecskemeti, C. Morin, L. Ricci, J. Sahuquillo, M. Schulz, V. Scarano, S. L. Scott, and J. Weidendorfer, Eds., vol. 8374. Springer, 2013, pp. 680–689. [Online]. Available: https://doi.org/10.1007/978-3-642-54420-0_66

[14] F. Fraternali, A. Bartolini, C. Cavazzoni, and L. Benini, "Quantifying the Impact of Variability and Heterogeneity on the Energy Efficiency for a Next-Generation Ultra-Green Supercomputer," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 7, pp. 1575–1588, 2018.

[15] Y. Jiao, H. Lin, P. Balaji, and W. Feng, "Power and Performance Characterization of Computational Kernels on the GPU," in *IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing*, 2010, pp. 221–228.

[16] T. Scogland, J. Azose, D. Rohr, S. Rivoire, N. Bates, and D. Hackenberg, "Node Variability in Large-Scale Power Measurements: Perspectives from the Green500, Top500 and EEHPCWG," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2807591.2807653

[17] J. Tan and K. Yan, "Efficiently Managing the Impact of Hardware Variability on GPUs' Streaming Processors," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 24, no. 1, dec 2018. [Online]. Available: https://doi.org/10.1145/3287308

[18] P. Sinha, A. Guliani, R. Jain, B. Tran, M. D. Sinclair, and S. Venkataraman, "Not All GPUs Are Created Equal: Characterizing Variability in Large-Scale, Accelerator-Rich Systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '22. IEEE Press, 2022.

[19] ——, "Not All GPUs Are Created Equal: Characterizing Variability in Large-Scale, Accelerator-Rich Systems," 2022. [Online]. Available: https://arxiv.org/abs/2208.11035

[20] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS-W*, 2017.

[21] TACC, "Texas Advanced Computing Center," https://www.tacc.utexas.edu/, 2024.

[22] W. Gao, Q. Hu, Z. Ye, P. Sun, X. Wang, Y. Luo, T. Zhang, and Y. Wen, "Deep Learning Workload Scheduling in GPU Datacenters: Taxonomy, Challenges and Vision," 2022.

[23] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A GPU Cluster Manager for Distributed Deep Learning," in *16th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI. Boston, MA: USENIX Association, Feb. 2019, pp. 485–500. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/gu

[24] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing, "Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning," in *15th USENIX Symposium on Operating Systems Design and Implementation*, ser. OSDI. USENIX Association, Jul. 2021, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/qiao

[25] D. Narayanan, F. Kazhamiaka, F. Abuzaid, P. Kraft, A. Agrawal, S. Kandula, S. Boyd, and M. Zaharia, "Solving Large-Scale Granular Resource Allocation Problems Efficiently with POP," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 521–537. [Online]. Available: https://doi.org/10.1145/3477132.3483588

[26] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads," in *14th USENIX Symposium on Operating Systems Design and Implementation*, ser. OSDI. USENIX Association, Nov. 2020, pp. 481–498. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/narayanan-deepak

[27] S. Agarwal, A. Phanishayee, and S. Venkataraman, "Blox: A Modular Toolkit for Deep Learning Schedulers," 2023.

[28] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective Cluster Scheduling for Deep Learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 595–610. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/xiao

[29] S. Jayaram Subramanya, D. Arfeen, S. Lin, A. Qiao, Z. Jia, and G. R. Ganger, "Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 642–657. [Online]. Available: https://doi.org/10.1145/3600006.3613175

[30] J. Mohan, A. Phanishayee, J. Kulkarni, and V. Chidambaram, "Looking Beyond GPUs for DNN Scheduling on Multi-Tenant Clusters," in *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*, M. K. Aguilera and H. Weatherspoon, Eds.

USENIX Association, 2022, pp. 579–596. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/mohan

[31] G. Ostrouchov, D. Maxwell, R. A. Ashraf, C. Engelmann, M. Shankar, and J. H. Rogers, "GPU Lifetimes on Titan Supercomputer: Survival Analysis and Reliability," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–14.

[32] A. Gholami, "AI and Memory Wall," 2021.

[33] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism," 2019.

[34] S. Naffziger, N. Beck, T. Burd, K. Lepak, G. H. Loh, M. Subramony, and S. White, "Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families : Industrial Product," in *ACM/IEEE 48th Annual International Symposium on Computer Architecture*, ser. ISCA, 2021, pp. 57–70.

[35] R. Adolf, S. Rama, B. Reagen, G. Wei, and D. Brooks, "Fathom: reference workloads for modern deep learning methods," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2016, pp. 1–10. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IISWC.2016.7581275

[36] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás, "DVFS-aware application classification to improve GPGPUs energy efficiency," *Parallel Computing*, vol. 83, pp. 93–117, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819118300243

[37] NVIDIA, "Nsight Compute Documentation," Accessed 2024. [Online]. Available: https://docs.nvidia.com/nsight-compute/NsightCompute/index.html

[38] P. J. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377042787901257

[39] Google, "gRPC: A high performance, open source universal RPC framework," Accessed 2024. [Online]. Available: https://grpc.io/

[40] M. Jeon, S. Venkataraman, A. Phanishayee, u. Qian, W. Xiao, and F. Yang, "Analysis of Large-scale Multi-tenant GPU Clusters for DNN Training Workloads," in *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '19. USA: USENIX Association, 2019, p. 947–960.

[41] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla, "Themis: Fair and Efficient GPU Cluster Scheduling," in *17th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 289–304. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/mahajan

[42] M. Amaral, J. Polo, D. Carrera, S. Seelam, and M. Steinder, "Topology-aware GPU scheduling for learning workloads in cloud environments," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126933

[43] A. Hankin, D. Werner, M. Amiraski, J. Sebot, K. Vaidyanathan, and M. Hempstead, "HotGauge: A Methodology for Characterizing Advanced Hotspots in Modern and Next Generation Processors," in *IEEE International Symposium on Workload Characterization*, ser. IISWC, 2021, pp. 163–175.

[44] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR, 2017, pp. 77–85.

[45] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," *CoRR*, vol. abs/1512.03012, 2015. [Online]. Available: http://arxiv.org/abs/1512.03012

[46] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[47] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[48] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1511.06434

[49] F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao, "LSUN: construction of a large-scale image dataset using deep learning with humans in the loop," *CoRR*, vol. abs/1506.03365, 2015. [Online]. Available: http://arxiv.org/abs/1506.03365

[50] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: https://doi.org/10.18653/v1/n19-1423

[51] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism," *CoRR*, vol. abs/1909.08053, 2019. [Online]. Available: http://arxiv.org/abs/1909.08053

[52] G. Attardi, "WikiExtractor," https://github.com/attardi/wikiextractor, 2015.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[54] "ResNet: Deep residual networks pre-trained on ImageNet," https://pytorch.org/hub/pytorch_vision_resnet/.

[55] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, ser. NAACL-HLT, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: https://doi.org/10.18653/v1/n19-1423

[56] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron, "Pannotia: Understanding Irregular GPGPU Graph Applications," in *IEEE International Symposium on Workload Characterization*, ser. IISWC, Sept 2013, pp. 185–195.

[57] NVIDIA, "System Management Interface SMI," Accessed 2024. [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface

[58] H. Menon, B. Acun, S. G. De Gonzalo, O. Sarood, and L. Kalé, "Thermal aware automated load balancing for HPC applications," in *IEEE International Conference on Cluster Computing*, ser. CLUSTER, 2013, pp. 1–8.

[59] B. Acun and L. V. Kale, "Mitigating Processor Variation through Dynamic Load Balancing," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 1073–1076.

[60] B. Acun, E. K. Lee, Y. Park, and L. V. Kale, "Support for Power Efficient Proactive Cooling Mechanisms," in *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*, 2017, pp. 94–103.

[61] D. Chasapis, M. Casas, M. Moretó, M. Schulz, E. Ayguadé, J. Labarta, and M. Valero, "Runtime-Guided Mitigation of Manufacturing Variability in Power-Constrained Multi-Socket NUMA Nodes," in *Proceedings of the 2016 International Conference on Supercomputing*, ser. ICS '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2925426.2926279

[62] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda, M. Kondo, and I. Miyoshi, "Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing," in *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–12.

[63] J. Tan, M. Chen, Y. Yi, and X. Fu, "Mitigating the Impact of Hardware Variability for GPGPUs Register File," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 11, pp. 3283–3297, 2016.

[64] D. S. Maura, T. Goel, K. Goswami, D. S. Banerjee, and S. Das, "Variation aware power management for GPU memories," *Microprocessors and Microsystems*, vol. 96, p. 104711, 2023.

[Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933122002411

[65] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60.

[66] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. 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*, ser. SC '21. ACM, Nov. 2021. [Online]. Available: http://dx.doi.org/10.1145/3458817.3476223