Analyzing GPU Utilization in HPC Workloads: Insights from Large-Scale Systems

Efe Sencan
Boston University
Boston, MA, USA
esencan@bu.edu

Dhruva Kulkarni
Lawrance Berkeley National Laboratory
Berkeley, CA, USA
dkulkarni@lbl.gov

Ayse Coskun
Boston University
Boston, MA, USA
acoskun@bu.edu

Kadidia Konate
Lawrance Berkeley National Laboratory
Berkeley, CA, USA
kadidiaKonate@lbl.gov

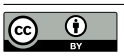
Abstract

Efficient resource utilization is a critical yet challenging problem in high-performance computing (HPC) systems, particularly in GPU-accelerated workloads. GPUs have become indispensable in modern datacenters, powering applications in AI, scientific computing, and large-scale simulations, but their potential is often hindered by imbalances in resource allocation and utilization. These inefficiencies across GPUs and nodes can lead to significant performance degradation and energy wastage. While existing works analyze resource utilization at a coarse level, they often fail to capture the intricate temporal and spatial imbalances present in multi-node GPU jobs. In this work, we analyze GPU jobs executed on the Perlmutter supercomputer, a GPU-accelerated system at the National Energy Research Scientific Computing Center (NERSC), using telemetry data from one month of operation in 2024. We identify inefficiencies in how resources are allocated and utilized across GPU nodes and GPUs within nodes. We propose novel methodologies to quantify these imbalances using refined metrics that capture both spatial and temporal variations in resource utilization. Our analysis reveals that GPU utilization is generally well-balanced temporally for most jobs, with no significant temporal imbalances observed. However, GPU memory utilization remains consistently low across many jobs, highlighting opportunities to optimize resource allocation and improve memory usage efficiency.

Keywords

Workload Analysis, Resource Utilization, HPC Systems, GPU

ACM Reference Format:
Efe Sencan, Dhruva Kulkarni, Ayse Coskun, and Kadidia Konate. 2025. Analyzing GPU Utilization in HPC Workloads: Insights from Large-Scale Systems. In *Practice and Experience in Advanced Research Computing (PEARC '25), July 20–24, 2025, Columbus, OH, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3708035.3736010



PEARC '25, Columbus, OH, USA

ACM ISBN 979-8-4007-1398-9/25/07
https://doi.org/10.1145/3708035.3736010

1 Introduction

High-Performance Computing (HPC) systems have become indispensable for accelerating scientific discovery and technological innovation, enabling large-scale simulations and data-intensive computations across disciplines such as astrophysics, genomics, climate modeling, and materials science. Over the last decade, the shift from traditional CPU-centric architectures toward heterogeneous systems integrating GPUs and other specialized accelerators has played a pivotal role in meeting the escalating demands of modern workloads, including machine learning (ML) and deep learning (DL) applications [7, 9, 12, 20].

Despite their growing ubiquity, GPU-accelerated systems pose unique challenges in effective resource allocation, scheduling, and performance optimization [11, 13, 25]. As job compositions become increasingly diverse—with various codes and libraries imposing heterogeneous demands on GPU memory, compute cores, and interconnect bandwidth—achieving high utilization across all resources becomes a non-trivial endeavor [17]. Even within a single node allocated to a single job, imbalances can arise among the multiple GPUs due to variations in parallel efficiency, communication overheads, and irregular memory access patterns [3, 8]. These internal imbalances can lead to suboptimal performance and resource underutilization, ultimately increasing both the runtime and the energy cost of the job.

Over the past decade, a broad range of workload characterization studies have aimed to illuminate inefficiencies in HPC environments. Early efforts largely focused on CPU-centric clusters, analyzing batch logs to reveal patterns in job runtimes, queue wait times, and memory usage. For example, Bang et al. [2] performed feature selection and clustering on HPC workloads to detect underutilized resources and frequent job patterns, while Patel et al. [21] conducted a long-term trend analysis on Argonne supercomputers (e.g., Intrepid, Mira) to show that mid-sized jobs gradually dominated the system's node hours, highlighting challenges in scheduling and provisioning for evolving CPU workloads. These works, however, primarily address CPU usage, memory capacity, and I/O usage without delving deeply into fine-grained GPU utilization or node-level GPU resource telemetry.

More recent research recognizes that GPU-accelerated HPC systems introduce additional complexities. For instance, Li et al. [16] examined the Perlmutter system at NERSC to show that both CPUs and GPUs were frequently underutilized, with many GPU-enabled

jobs consuming less than half of the available memory capacity. Similarly, Shin et al. [22] studied a pre-exascale GPU cluster, focusing on node-level power and thermal metrics. They uncovered substantial temperature-induced performance bottlenecks and partial GPU utilization, demonstrating that new instrumentation (e.g., power monitors) can yield deeper insights into job inefficiencies. Yet, such studies often present GPU usage data at a node-level granularity, leaving per-GPU temporal load imbalances underexplored. Additionally, newly emerging HPC+ML workloads pose fresh scheduling challenges—jobs may be short but highly GPU-intensive, or exhibit large memory footprints with sporadic usage patterns [6].

To address these gaps, we combine job-level metadata (e.g., Slurm logs with queue times, node allocations) and time-series GPU telemetry from NVIDIA's Data Center GPU Manager (DCGM) [19]. This strategy enables a finer-grained analysis of how multiple GPUs within the same node are actually utilized over time, highlighting potential spatial imbalance (uneven load distribution across GPUs) and temporal imbalance (fluctuating usage within each GPU over the job's runtime). This analysis uncovers inefficiencies in GPU resource utilization, providing a foundation for future scheduling improvements, such as better load balancing across GPUs, dynamic resource scaling, and optimized backfilling policies, to enhance performance and energy efficiency in modern HPC environments. Our contributions are as follows:

- We analyze detailed DCGM telemetry data to study GPU resource utilization in large-scale HPC workloads at NERSC.
- We introduce refined metrics for quantifying spatial and temporal imbalances in GPU workloads, capturing intra- and inter-node utilization disparities.
- We provide a systematic approach for measuring imbalance factors, offering a detailed assessment of workload inefficiencies in production supercomputing environments.

In the remainder of this paper, we discuss how our methods bridge the gap between coarse system-wide measurements and the fine-grained view required for truly effective resource management. Although our focus is on Perlmutter, the methodologies and observations generalize to other heterogeneous HPC environments seeking to optimize GPU utilization, job throughput, and overall energy efficiency.

2 Related Work

Researchers characterize HPC I/O from various angles, initially focusing on simulation-based workloads but now expanding to include deep learning (DL) and machine learning (ML). Early efforts develop lightweight logging frameworks (e.g., Darshan) to correlate application metrics with system logs for I/O performance anomaly detection [4]. These studies reveal that while checkpoint/restart often dominates simulation workloads, ML workloads exhibit diverse patterns, particularly read-intensive or small-batch writes [14]. Additionally, studies show that ML and mixed workloads in HPC centers frequently underutilize node resources, motivating more adaptive resource-sharing approaches [16].

Several projects propose scheduling policies for heterogeneous workloads, recognizing that uniform time-sharing often underutilizes GPUs if applications fail to saturate hardware [10, 18]. Historically, GPU-sharing research focuses on time-slicing or best-effort space sharing, where standard approaches (e.g., MPS) risk interference [24]. Recently, NVIDIA's Multi-Instance GPU (MIG) introduced compute and memory isolation, which prevents cross-application interference and improves utilization. However, slicing constraints may degrade single-job throughput or increase energy consumption [15].

These trade-offs align with broader trends in resource disaggregation, where partitioning memory or hardware aims to boost cluster efficiency [18]. However, standard approaches often neglect ML-driven HPC usage, where datasets and models introduce complex read–write patterns. Current studies examine how GPU partitioning complements I/O optimizations, staging policies, and scheduling heuristics for emerging AI workflows [14, 16].

Ephemeral HPC resource provisioning is another strategy for handling ML workloads with fluctuating hardware demands. Dynamic resource allocation for large-scale training and concurrency scheduling boosts throughput by adapting to job phases [1, 23]. Such efforts emphasize the importance of flexible resource provisioning alongside robust I/O strategies. Our study complements these directions by analyzing GPU resource utilization and performance inefficiencies across representative workloads, offering insights to inform dynamic scheduling strategies.

3 Methodology

To analyze GPU resource utilization patterns and identify inefficiencies in HPC workloads, we integrate time-series telemetry from the DCGM with job-level metadata from the Slurm scheduler. This combined dataset enables a detailed assessment of workload behavior across both spatial and temporal dimensions.

3.1 Dataset Overview

Our study examines GPU workloads executed on Perlmutter during July 2024. Perlmutter consists of 1,536 GPU-accelerated nodes, each equipped with four NVIDIA A100 GPUs with 40 GB of HBM2 memory. These GPUs are interconnected within nodes using NVIDIA NVLink and across nodes via HPE's Cray Slingshot network. The dataset includes DCGM telemetry, which provides periodic snapshots of GPU activity, such as utilization, memory activity, power consumption, and interconnect traffic. These metrics are sampled at 10-second intervals during job execution. We also integrate Slurm job metadata, capturing job submission and completion times, runtime durations, allocated resources, and job exit statuses. To focus on relevant workloads, we filter the dataset to include only GPU jobs submitted under the regular Quality of Service (QoS) category, resulting in 118,276 jobs. This subset represents only a fraction of the total jobs executed on Perlmutter during the study period and may not fully capture the entire workload landscape.

Each job may allocate one or more GPU nodes, where each node contains four GPUs. Within a node, each GPU is identified by a unique `gpu_id` ranging from 0 to 3. For multi-node jobs, telemetry streams are recorded separately for each ⟨node, `gpu_id`⟩ pair. The telemetry data for a single GPU is represented as:

$$X_{j,n,g} \in \mathbb{R}^{M \times T} \tag{1}$$

where j denotes the job identifier, n represents the node, and g is the GPU identifier. Here, M corresponds to the number of telemetry metrics, and T represents the job duration in seconds, which determines the time series length. If a job terminates early, telemetry records exist only up to that point.

3.2 DCGM Metrics

DCGM provides fine-grained telemetry for each GPU, capturing compute activity, memory performance, network traffic, and power consumption. Table 1 summarizes the key metrics used in our analysis.

3.3 Feature Extraction

To extract essential time-series characteristics, we use the TSFRESH [5] package. TSFRESH automates the extraction of a diverse set of features from time-series data, enabling detailed workload analysis. Specifically, we extract 17 features for each DCGM metric, resulting in a total of 408 feature columns after processing. These features include standard summary statistics such as mean, standard deviation, and coefficient of variation, which provide insights into the central tendency and variability of utilization metrics. Additionally, we compute the linear trend of GPU activity over time to capture overall patterns of increasing or decreasing usage. The mean absolute change (MAC) quantifies fluctuations between consecutive measurements, offering a finer-grained understanding of temporal variability. These features help identify workload characteristics and potential inefficiencies in resource utilization, serving as a foundation for further analysis. After feature extraction, we represent each job as a set of feature vectors, where each vector corresponds to a physical GPU allocated to that job across all assigned GPU nodes. Specifically, for a job j running on node n and utilizing GPU g, we define the extracted feature representation as:

$$f_{j,n,g} \in \mathbb{R}^{K} \tag{2}$$

where K is the total number of extracted features per GPU. This representation reduces data volume while preserving essential workload characteristics. We store the processed data in Parquet format for efficient storage and retrieval in our analysis pipeline.

4 Dataset Characteristics Analysis

To evaluate peak resource demands and utilization in GPU workloads, we analyze the maximum values of key telemetry metrics, including GPU utilization, memory utilization, and power consumption. These peak values provide insights into workload intensity and variations in resource allocation efficiency. We integrate job-level metadata from Slurm, including job duration and queue wait time, to analyze scheduling and execution characteristics. This integration ensures an accurate linkage between jobs and their resource usage, providing a comprehensive evaluation of system performance. Table 2 presents summary statistics for all jobs, capturing peak GPU utilization, memory utilization, and power consumption at any point during execution. On average, jobs exhibit a GPU utilization of 71.77%, while memory utilization remains significantly lower at 28.64%, indicating that many workloads underutilize available memory resources. GPU memory copy utilization also varies widely, with a mean of 41.98% and a standard deviation of 40.01%, indicating significant variability in the utilization of the GPU's dedicated copy engine (DMA) for memory transfer operations across workloads.

We divide jobs into groups based on their duration and the number of allocated nodes, as detailed in Tables 3a and 3b. Table 3a shows that 78.30% of jobs run for less than one hour, and 93.42% complete within six hours. Only 2.60% of jobs exceed 12 hours; however, these longer jobs account for 27.14% of the total GPU node hours, highlighting their significant contribution to overall resource consumption despite their lower frequency in the workload. Table 3b highlights that 80.12% of jobs allocate a single GPU node, while jobs involving more than 128 nodes account for only 0.15% of the total.

5 Resource Utilization of Jobs

Figures 1 (a) and (b) illustrate the distribution of GPU utilization and GPU memory utilization across all jobs, providing insights into resource allocation efficiency. The utilization bins represent the maximum utilization reached at any point during job execution. For instance, if a job falls into the (0, 15]% utilization bin, it means the job never exceeded 15% utilization throughout its runtime.

The majority of jobs (62.75%) reach at least 75% GPU utilization at some point, contributing to 80.01% of total node hours, suggesting that most jobs efficiently utilize GPU compute resources. Although 14.11% of jobs remain at 0% GPU utilization, these jobs contribute to only 3.06% of total node hours, thus the overall impact on system-wide resource consumption is limited.

GPU memory utilization remains significantly lower across many workloads compared to GPU compute utilization. Only 15.74% of jobs reach 75% or higher memory utilization, contributing to 20.40% of total node hours. In contrast, 37.12% of jobs never exceed 15% memory utilization, accounting for 37.03% of total node hours, highlighting widespread underutilization of available GPU memory. 8.85% of jobs reports 0% GPU memory utilization, meaning they never engage GPU memory during execution. However, these jobs account for only 0.32% of total node hours, indicating that their impact on overall resource consumption is minimal. The presence of fully idle GPUs suggests cases where jobs allocate GPU resources but fail to use them effectively, potentially due to workload misconfiguration or application characteristics that do not require significant memory usage. While these idle cases affect only a small fraction of node hours, the broader trend of low memory utilization presents an opportunity for optimizing workload placement and refining memory allocation strategies.

6 Quantifying Temporal and Spatial Imbalance in GPU Workloads

Analyzing resource utilization in HPC workloads requires not only measuring overall GPU usage but also understanding how resources fluctuate over time and across allocated GPUs. Imbalanced utilization can lead to inefficient workload execution, resource contention, and prolonged queue wait times. To address these inefficiencies, we introduce *temporal imbalance factors*, which measure variations

Table 1: Key DCGM metrics used in this study.

Metric Name	Description
`dram_active`	Percentage of time GPU's HBM2 memory is actively serving read/write requests.
`sm_active`	Fraction of cycles during which at least one warp was active on any Streaming Multiprocessor.
`tensor_active`	Fraction of cycles during which the tensor core is executing tensor operations.
`gpu_utilization`	Overall GPU utilization percentage, including SM activity and memory transfers.
`fb_free`	Free frame buffer memory, in MB, representing available GPU DRAM.
`fb_used`	Amount of frame buffer memory currently in use (MB) by the GPU.
`memory_utilization`	Percentage of GPU DRAM used: $\frac{\texttt{fb\_used}}{\texttt{fb\_used}+\texttt{fb\_free}} \times 100$
`NVLink Traffic`	Total data transferred over NVLink, measuring intra-node GPU communication (bytes/sec).
`power_usage`	GPU power consumption in watts.
`mem_copy_utilization`	Utilization of the GPU copy engine (i.e., fraction of time the dedicated memory copy units are active)

Table 2: Summary Statistics for All Jobs

Metric	Mean	Std	Min	Max
Job Duration (hrs)	1.64	3.70	0.00	24.00
Queue Wait Time (hrs)	0.04	0.07	0.00	1.88
GPU Utilization (%)	71.77	37.76	0.00	100.00
GPU Memory Utilization (%)	28.64	30.27	0.00	100.00
GPU Memory Copy Utilization (%)	41.98	40.01	0.00	100.00
Power Usage (W)	244.32	134.11	48.99	604.44
Allocated Nodes	3.14	16.88	1.00	1536

Table 3: Job Duration and GPU Node Allocation Distribution with GPU Node Hours

(a) Job Duration Distribution

Duration (hrs)	Jobs (%)	GPU Node Hours (%)
(0, 1]	78.30	12.24
(1, 3]	11.32	20.54
(3, 6]	3.80	15.06
(6, 12]	3.98	25.03
(12, 24]	2.60	27.14

(b) GPU Node Allocation Distribution

GPU Nodes	Jobs (%)	GPU Node Hours (%)
[1]	80.12	17.80
(1, 4]	13.28	21.35
(4, 16]	4.33	22.54
(16, 64]	1.85	21.59
(64, 128]	0.28	9.58
(128, 1536]	0.15	7.14

in GPU utilization over the course of a job, and *spatial imbalance factors*, which quantify disparities in utilization across GPUs allocated to the same job. These metrics provide insights into workload execution patterns, helping identify opportunities for improved scheduling and resource allocation.

6.1 Temporal Imbalance Factor Calculation

To quantify *temporal imbalance*, we explore multiple formulations that leverage feature-extracted representations of GPU resource utilization.

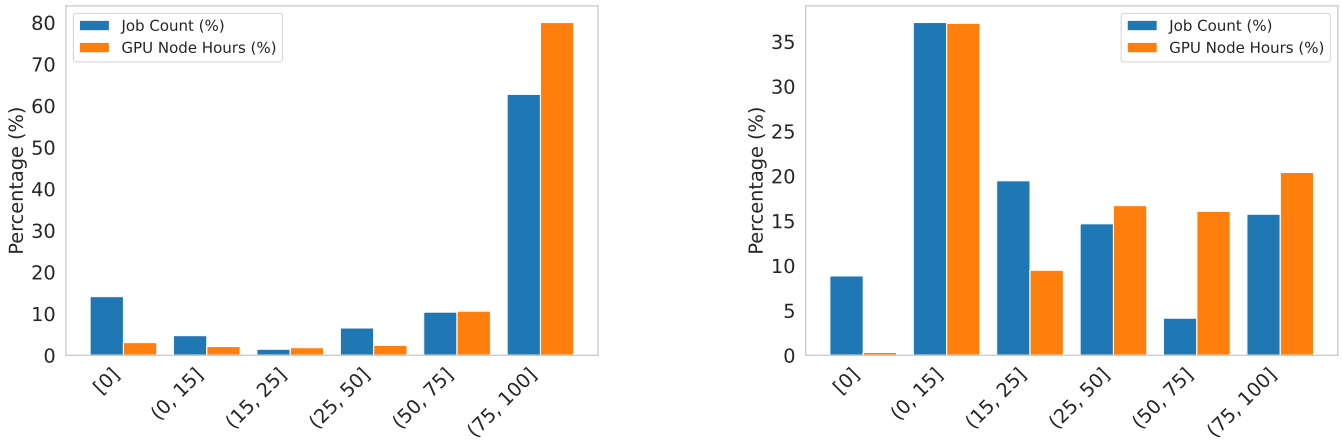
One approach is the *coefficient of variation (CV)*, which measures the standard deviation relative to the mean utilization over time for each GPU. This metric captures fluctuations in resource usage, where a higher CV value indicates greater variation and less consistent utilization. For jobs that span multiple GPUs across nodes, the job-level temporal imbalance factor is computed by first calculating the CV for each GPU and then selecting the highest CV value across all GPUs and nodes allocated to the job:

$$\text{Temporal Imbalance Factor (CV)} = \max_{g \in G, n \in N} \left(\frac{\sigma(U_{g,n})}{\mu(U_{g,n})} \right), \quad (3)$$

where $U_{g,n}$ represents the time-series utilization of GPU g on node n, $\sigma(U_{g,n})$ is the standard deviation, and $\mu(U_{g,n})$ is the mean utilization.

Another method involves analyzing the linear trend of utilization over time using linear regression. The slope of the fitted regression line indicates whether utilization increases or decreases, helping quantify variations in workload execution. A steep positive or negative slope suggests dynamic workload phases with significant variations in GPU usage. For jobs running on multiple GPUs and nodes, the job-level temporal imbalance factor is determined by selecting the maximum trend value across all GPUs and nodes:

$$\text{Temporal Imbalance Factor (Trend)} = \max_{g \in G, n \in N} \left(\left| \text{slope}(U_{g,n}) \right| \right). \quad (4)$$



(a) GPU Utilization (%) **(b) GPU Memory Utilization (%)**

Figure 1: Distribution of jobs and node hours for GPU utilization and GPU memory utilization.

Additionally, we consider the MAC, which captures the average change between consecutive utilization values, providing finer-grained insights into GPU utilization fluctuations. Similar to the other metrics, the maximum MAC value across all GPUs and nodes is used to compute the job-level temporal imbalance factor:

$$\text{Temporal Imbalance Factor (MAC)} = \max_{g\in G, n\in N}\left(\frac{1}{T}\sum_{t=1}^{T-1}|U_{g,n}(t+1) - U_{g,n}(t)|\right). \tag{5}$$

Each of these formulations captures distinct aspects of temporal imbalance, offering a comprehensive perspective on workload execution patterns.

To provide a holistic measure of temporal imbalance, we combine the three individual metrics—*CV*, *Trend*, and *MAC*—into a single *merged temporal imbalance factor*. Each metric is first normalized to the [0, 1] range using Min-Max scaling to ensure that differences in scale do not distort the final result. We assign a lower weight to the *Trend* metric because it often introduces noise when fluctuations obscure clear increasing or decreasing patterns. To mitigate this, we emphasize *CV* and *MAC*, which more directly capture utilization variability. The merged temporal imbalance factor is computed as:

$$\begin{aligned}\text{Merged TIF} = 0.4 \times \text{Normalized CV} + 0.4 \times \text{Normalized MAC} + \\ + 0.2 \times \text{Normalized Trend}.\end{aligned} \tag{6}$$

This formulation ensures that temporal imbalance is calculated across all GPUs and nodes for each job, prioritizing short-term fluctuations and overall variability while maintaining the trend component as a secondary contributor. Figure 2 illustrates the temporal imbalance factors for GPU and memory utilization. The plot for GPU utilization (Figure 2a) demonstrates that most jobs exhibit low temporal imbalance, as evidenced by the sharp rise in the CDF, indicating consistent usage patterns across the majority of workloads. Memory utilization (Figure 2b) shows a similar trend, with low temporal imbalance observed for the majority of jobs. While some jobs exhibit slightly elevated temporal imbalance values, these represent a small fraction of the total workload and do not indicate a significant long tail effect. Additionally, analyses of other GPU resource metrics, including `sm_active`, `tensor_active`, and `dram_active`, reveal similar patterns. Temporal imbalance remains low across the majority of jobs for these metrics, suggesting balanced resource usage over time in most cases. However, a subset of jobs with higher temporal imbalance may indicate irregular or fluctuating workload behavior, warranting further exploration to identify underlying causes or specific workload characteristics.

6.2 Spatial Imbalance Factor Calculation

While temporal imbalance reflects changes in GPU utilization over time, *spatial imbalance* measures how evenly GPU utilization is distributed across allocated resources both within and across nodes. We define two types of spatial imbalance:

Intra-Node Imbalance: Intra-node imbalance quantifies GPU utilization disparities among GPUs within the same node. To capture this, we compute the maximum intra-node imbalance across all nodes allocated to a job. The first metric, the normalized range (NR), is defined as:

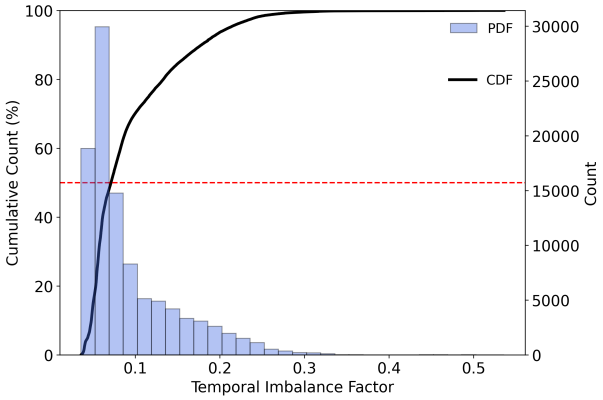
$$\text{Intra-Node Spatial Imbalance (NR)} = \max_{n\in N}\frac{\max_{g\in G_n} U_g - \min_{g\in G_n} U_g}{\max_{g\in G_n} U_g}, \tag{7}$$

where G_n represents the set of GPUs within node n, and U_g is the utilization of GPU g. This ensures that for jobs spanning multiple nodes, we retain the maximum observed intra-node imbalance.

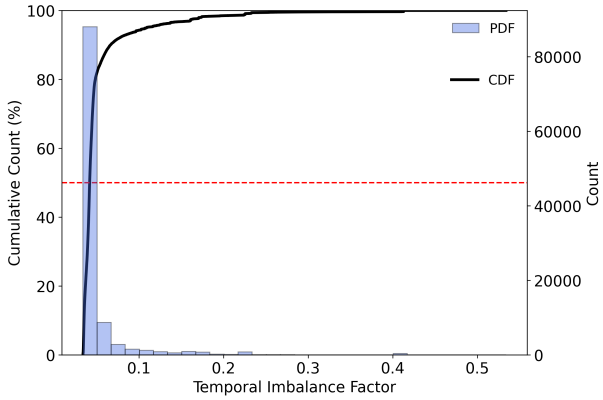
We also compute the maximum variance of GPU utilization within nodes for each job:

$$\text{Intra-Node Spatial Imbalance (Variance)} = \max_{n\in N}\sigma^2(U_g) \quad \forall g \in G_n. \tag{8}$$

Inter-Node Imbalance: Inter-node imbalance captures disparities in GPU utilization across multiple nodes allocated to a job. Similar to intra-node imbalance, we retain the maximum inter-node imbalance observed among the nodes allocated to each job. The normalized range (NR) across nodes is defined as:



(a) GPU Utilization



(b) Memory Utilization

Figure 2: CDF and PDF plots of temporal imbalance factors for various GPU metrics. The plots illustrate the distribution of imbalance levels across jobs, emphasizing variations in GPU and memory utilization.

$$\text{Inter-Node Spatial Imbalance (NR)} = \frac{\max_{n \in N} U_n - \min_{n \in N} U_n}{\max_{n \in N} U_n}, \tag{9}$$

where U_n is the average GPU utilization of node n, and N is the set of nodes allocated to job j.

Similarly, we compute the maximum variance of GPU utilization across nodes for each job:

$$\text{Inter-Node Spatial Imbalance (Variance)} = \sigma^2(U_n) \quad \forall n \in N. \tag{10}$$

To provide a holistic perspective on workload distribution, we compute merged spatial imbalance factors by averaging the normalized range and variance metrics for both intra-node and inter-node imbalances:

$$\text{Merged Intra-Node Spatial Imbalance} = \frac{\text{Intra-Node NR} + \text{Intra-Node Variance}}{2}. \tag{11}$$

$$\text{Merged Inter-Node Spatial Imbalance} = \frac{\text{Inter-Node NR} + \text{Inter-Node Variance}}{2}. \tag{12}$$

These formulations ensure that the most significant spatial imbalance observed within or across nodes is retained, providing a more accurate characterization of workload distribution inefficiencies.

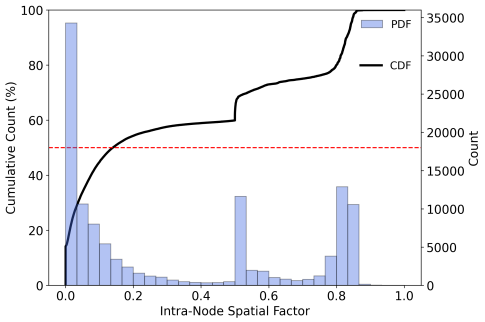
Figure 3 visualizes the cumulative distribution function (CDF) and probability density function (PDF) for the merged intra-node and inter-node spatial imbalance factors for selected GPU utilization metrics. The CDFs demonstrate that the majority of jobs exhibit low intra-node and inter-node spatial imbalance values, indicating balanced resource usage in most cases. However, the right tails of the PDFs highlight the presence of outliers, where substantial disparities exist in resource usage either within or across nodes. Notably, GPU utilization imbalance tends to be higher compared to memory utilization imbalance, likely due to varied workload characteristics and differences in computation intensity or GPU affinity within nodes. Furthermore, intra-node imbalance is generally more pronounced than inter-node imbalance, indicating that disparities in resource usage are more common among GPUs within the same node than across nodes.

6.3 Correlation Analysis of Temporal and Spatial Imbalances

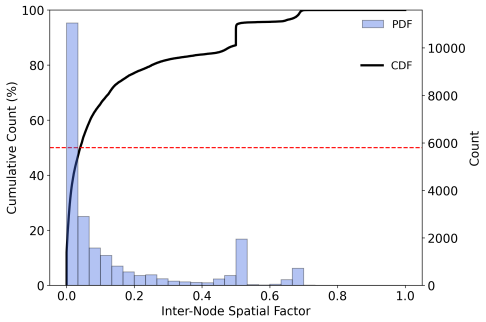
The correlation matrix in Figure 4 provides a quantitative analysis of the relationships between temporal imbalance, spatial imbalance (intra-node and inter-node), resource utilization metrics, and job-level characteristics such as node hours. A key observation is the moderate negative correlation (-0.42) between temporal imbalance in memory utilization and maximum GPU utilization, suggesting that jobs with stable memory usage tend to achieve better GPU efficiency. This insight implies that improving memory utilization stability could enhance GPU performance for certain workloads. Furthermore, intra-node and inter-node spatial imbalances for both GPU and memory utilization are strongly correlated (0.89 for intra-node and 0.55 for inter-node), indicating that resource disparities within nodes often extend across nodes. Node hours show weak or negligible correlations with imbalance factors and utilization metrics, revealing that larger jobs (in terms of node usage) are not inherently more or less imbalanced, pointing to other workload-level factors influencing imbalance. These findings highlight the potential for actionable improvements in resource allocation and scheduling strategies. For instance, jobs with high temporal memory imbalance or significant spatial imbalance could be prioritized for improved placement or workload consolidation to reduce inefficiencies. Addressing intra-node imbalances through fine-grained scheduling could also mitigate cascading effects that exacerbate inter-node disparities. Additionally, further investigation into workload-specific patterns of imbalance could inform strategies to dynamically adjust resource allocations, ultimately enhancing overall system efficiency and throughput for GPU-accelerated HPC systems.

7 Conclusion

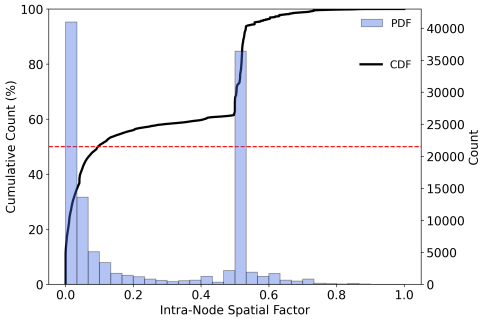
This work analyzes GPU resource utilization in HPC workloads on the Perlmutter supercomputer using DCGM telemetry and Slurm



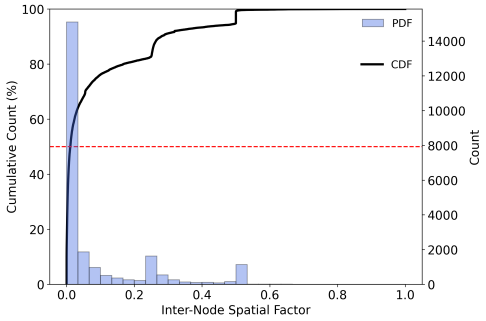
(a) Intra-Node GPU Utilization



(b) Inter-Node GPU Utilization



(c) Intra-Node Memory Utilization



(d) Inter-Node Memory Utilization

Figure 3: CDF and PDF plots of merged intra-node and inter-node spatial imbalance factors for GPU and memory utilization metrics.

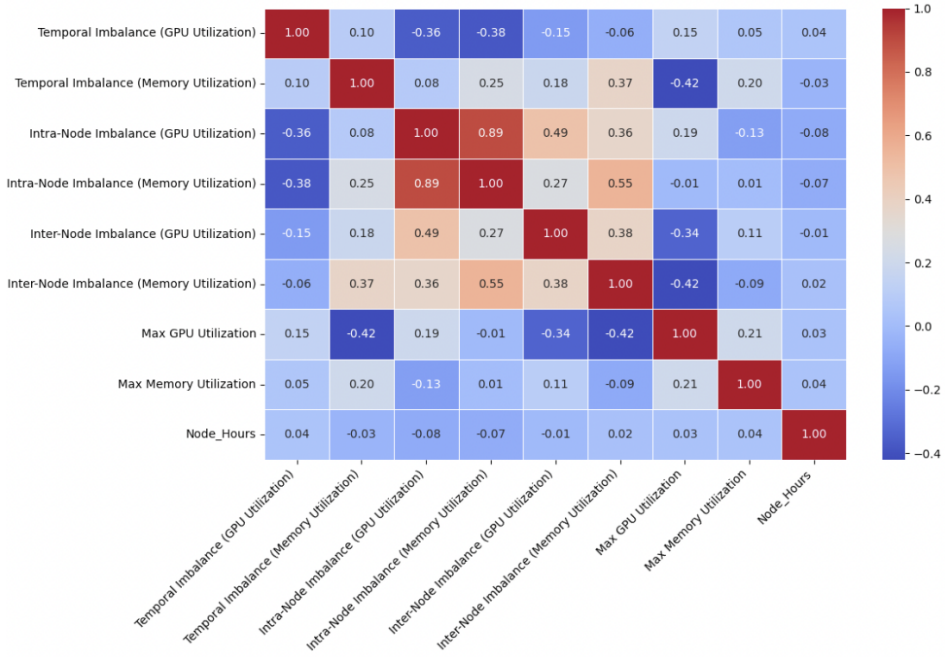


Figure 4: Correlation matrix showing relationships between temporal and spatial imbalance factors, maximum GPU and memory utilization, and node hours.

metadata, introducing refined metrics to quantify temporal and spatial imbalances and identify workload inefficiencies. Our analysis reveals that GPU compute utilization is generally well-balanced across most jobs, while memory utilization remains consistently low in over 50% of workloads, highlighting opportunities to better understand workload memory requirements and optimize resource provisioning strategies. Temporal imbalances are minimal for the majority of jobs, although outliers with significant fluctuations suggest that dynamic scheduling or workload adjustments could enhance performance. Spatial imbalances expose disparities in GPU usage both within and across nodes, with GPU utilization imbalance being more pronounced than memory utilization.

While this study highlights the prevalence of temporal and spatial imbalances, a quantitative assessment of their impact on system performance remains a key area for future research. Addressing these imbalances may require strategies such as improved scheduling policies, dynamic resource allocation, and targeted workload profiling. Future work will expand the scope to analyze temporal and spatial imbalances across a broader range of GPU workloads, assessing their impact on system-level performance. Additionally, we aim to explore the prerequisites, trade-offs, and practical strategies to mitigate imbalances, including how these interventions could enhance efficiency and resource utilization. This would provide a clearer understanding of the actionable steps needed to optimize GPU-accelerated HPC systems and quantify the performance improvements achievable through such efforts.

Acknowledgments

This research was supported by the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

References

[1] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2020. Batch: Machine learning inference serving on serverless platforms with adaptive batching. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[2] Jiwoo Bang, Chungyong Kim, Kesheng Wu, Alex Sim, Suren Byna, Sunggon Kim, and Hyeonsang Eom. 2020. HPC workload characterization using feature selection and clustering. In *Proceedings of the 3rd International Workshop on Systems and Network Telemetry and Analytics*. 33–40.

[3] Tal Ben-Nun, Michael Sutton, Sreepathi Pai, and Keshav Pingali. 2020. Groute: Asynchronous multi-GPU programming model with applications to large-scale graph processing. *ACM Transactions on Parallel Computing (TOPC)* 7, 3 (2020), 1–27.

[4] Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 2009. 24/7 characterization of petascale I/O workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 1–10.

[5] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. 2018. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing* 307 (2018), 72–77.

[6] Xiaoyu Chu, Daniel Hofstätter, Shashikant Ilager, Sacheendra Talluri, Duncan Kampert, Damian Podareanu, Dmitry Duplyakin, Ivona Brandic, and Alexandru Iosup. 2024. Generic and ML Workloads in an HPC Datacenter: Node Energy, Job Failures, and Node-Job Analysis. In *2024 IEEE 30th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 710–719.

[7] William J Dally, Stephen W Keckler, and David B Kirk. 2021. Evolution of the graphics processing unit (GPU). *IEEE Micro* 41, 6 (2021), 42–51.

[8] Debashis Ganguly, Ziyu Zhang, Jun Yang, and Rami Melhem. 2020. Adaptive page migration for irregular data-intensive applications under gpu memory oversubscription. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 451–461.

[9] Eugenio Gianniti, Li Zhang, and Danilo Ardagna. 2018. Performance prediction of gpu-based deep learning applications. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 167–170.

[10] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A {GPU} cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 485–500.

[11] Rong Gu, Yuquan Chen, Shuai Liu, Haipeng Dai, Guihai Chen, Kai Zhang, Yang Che, and Yihua Huang. 2021. Liquid: Intelligent resource estimation and network-efficient scheduling for deep learning jobs on distributed GPU clusters. *IEEE Transactions on Parallel and Distributed Systems* 33, 11 (2021), 2808–2820.

[12] Bagus Hanindhito and Lizy K John. 2024. Accelerating ml workloads using gpu tensor cores: The good, the bad, and the ugly. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*. 178–189.

[13] Pieter Hijma, Stijn Heldens, Alessio Sclocco, Ben Van Werkhoven, and Henri E Bal. 2023. Optimization techniques for GPU programming. *Comput. Surveys* 55, 11 (2023), 1–81.

[14] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. 2021. 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*. 1–15.

[15] Baolin Li, Viiay Gadepally, Siddharth Samsi, and Devesh Tiwari. 2022. Characterizing multi-instance gpu for machine learning workloads. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 724–731.

[16] Jie Li, George Michelogiannakis, Brandon Cook, Dulanya Cooray, and Yong Chen. 2023. Analyzing resource utilization in an HPC system: A case study of NERSC's Perlmutter. In *Proceedings of the International Conference on High Performance Computing*. Springer, 297–316.

[17] Zhongjin Li, Victor Chang, Haiyang Hu, Maozhong Fu, Jidong Ge, and Francesco Piccialli. 2021. Optimizing makespan and resource utilization for multi-DNN training in GPU cluster. *Future Generation Computer Systems* 125 (2021), 206–220.

[18] George Michelogiannakis, Benjamin Klenk, Brandon Cook, Min Yee Teh, Madeleine Glick, Larry Dennison, Keren Bergman, and John Shalf. 2022. A case for intra-rack resource disaggregation in HPC. *ACM Transactions on Architecture and Code Optimization (TACO)* 19, 2 (2022), 1–26.

[19] NVIDIA. [n. d.]. Data Center GPU Manager (DCGM). https://developer.nvidia.com/dcgm. Accessed 20 July 2021.

[20] Mohit Pandey, Michael Fernandez, Francesco Gentile, Olexandr Isayev, Alexander Tropsha, Abraham C Stern, and Artem Cherkasov. 2022. The transformational role of GPU computing and deep learning in drug discovery. *Nature Machine Intelligence* 4, 3 (2022), 211–221.

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

[22] Woong Shin, Vladyslav Oles, Ahmad Maroof Karimi, J Austin Ellis, and Feiyi Wang. 2021. Revealing power, energy and thermal dynamics of a 200pf pre-exascale supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[23] Teng Wang, Kathryn Mohror, Adam Moody, Kento Sato, and Weikuan Yu. 2016. An ephemeral burst-buffer file system for scientific applications. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 807–818.

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

[25] Zhisheng Ye, Wei Gao, Qinghao Hu, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. 2024. Deep learning workload scheduling in gpu datacenters: A survey. *Comput. Surveys* 56, 6 (2024), 1–38.