Plenty of Room in the Middle: Revealing Opportunities on a GPU Cluster via Fine-Grained Monitoring

ANDREW SHEINBERG*, SAMYAK GUPTA*, and KAI LI, Princeton University, U.S.A.

GPU clusters have become essential computing resources for academic and industrial research. However, their high cost and growing demand often result in long queueing delays. While previous monitoring efforts have operated at second- or minute-level frequencies, they primarily informed job-level scheduling for clusters and lacked the detail needed to optimize GPU-level utilization.

This paper presents an in-depth study of a university research GPU cluster, analyzing resource usage at 100ms intervals, aligning with modern OS preemptive scheduling time slices. We have developed a light-weight monitoring tool that leverages Nvidia's DCGM library [Nvidia 2020c] to capture fine-grained "compute utilization" metrics within GPUs. Deployed on a cluster with 89 nodes and 319 GPUs, our tool collected data over several months. This paper focuses on a 10-day period with 7,399 jobs submitted by 150 users from 12 departments. Our study reveals three key findings. (1) the GPU cluster exhibits significant underutilization, with over 70% of jobs using less than 20% of GPU memory and 50% averaging below 5% Streaming Multiprocessor (SM) utilization; (2) 80% of jobs have a drastic mismatch between their requested and actual durations, significantly limiting the scheduler's effectiveness; and (3) multi-GPU jobs show highly periodic communication patterns. These findings can help system designers to rethink scheduling and management strategies, potentially transforming the GPU systems software stack and the management of GPU clusters.

Our monitoring solution is available as open-source software at: https://github.com/als244/gpu_server_monitoring

CCS Concepts: • **Computer systems organization** → **Parallel architectures**; **Distributed architectures**; • **Hardware** → *Communication hardware, interfaces and storage.*

Additional Key Words and Phrases: GPU monitoring, GPU utilization, cluster scheduling, multiplexing

1 Introduction

GPU clusters have become the engines of modern AI, powering services and research across diverse domains. The insatiable appetite of large language models and the burgeoning demands of AI applications in science and other fields are driving an exponential growth in computing requirements. While GPU vendors continue to deliver increasingly powerful hardware with impressive performance, the associated costs have skyrocketed. For instance, Nvidia's latest Blackwell GPUs command a hefty price tag of $30,000–$40,000 [Haddad 2024]. However, GPU clusters are significantly underutilized [Hu et al. 2021; Jeon et al. 2019; Xiao et al. 2020], raising the critical question: How can we substantially improve GPU cluster utilization?

In typical GPU clusters, job schedulers assign jobs to GPUs without multitasking, with each GPU running a single job or task at a time. This approach persists even when a job uses only a small fraction of the GPU's Streaming Multiprocessors (SMs) or memory. The inefficiency is reminiscent of the early days of mainframe computers in the 1960s, where jobs were processed sequentially, leading to poor utilization of expensive hardware. The introduction of time-sharing operating systems [Corbató et al. 1962; Corbató et al. 1971] addressed this issue by enabling multiple jobs to share hardware resources efficiently. Yet, GPU clusters today remain constrained by similar inefficiencies.

*Both authors contributed equally to this research.

Authors' Contact Information: Andrew Sheinberg, asheinb@princeton.edu; Samyak Gupta, samyakg@princeton.edu; Kai Li, li@cs.princeton.edu, Princeton University, Department of Computer Science, Princeton, New Jersey, U.S.A..

Why do GPU clusters still rely on job schedulers without multitasking? Three key factors contribute to this limitation:

- Traditional GPUs do not support multitasking as they are designed for graphics [Adriaens et al. 2012].
- GPU's High-Bandwidth Memory (HBM) is fast but expensive and limited in capacity. Unlike CPUs, traditional GPUs lack mechanisms like virtual memory to allow multiple processes to share physical memory in a secure fashion.
- Parallel jobs are highly sensitive to task synchronization, where delays in one task can bottleneck the entire job, necessitating gang scheduling [Ousterhout 1982].

Recent hardware advances have begun addressing these limitations [Zhao et al. 2022]. Nvidia's Multi-Instance GPU (MIG) [Nvidia 2020a] and Multi-Process Service (MPS) [Nvidia 2020b] now enable concurrent task execution, while expanded memory capacities (up to 192GB on Blackwell) and Unified Virtual Memory (UVM) reduce memory constraints. These developments create new opportunities for multitasking-based utilization improvements.

To identify these opportunities, it is critical to understand the characteristics of current GPU jobs and identify opportunities for multitasking. Previous monitoring studies have relied on coarse-grained metrics, typically with sampling intervals of seconds to minutes, providing limited insights into the fine-grained utilization patterns of GPU resources. To solve this problem, we have developed a fine-grained monitoring tool capturing utilization data at 100ms intervals – aligning with modern operating system time slices. We have deployed this lightweight, scalable tool across a university research cluster of 89 nodes and 319 GPUs for several months.

This paper analyzes data collected during a 10-day observation period, encompassing 7,399 jobs submitted by 150 users across 12 departments. The analysis reveals three key findings:

- **Significant underutilization**: Over 70% of jobs used less than 20% of GPU memory, and 50% averaged below 5% SM utilization. This underutilization highlights a substantial gap between the capabilities of modern GPUs and the actual resource demands of many workloads, leaving much of the cluster's computational power untapped.
- **Pervasive mismatch between requested and actual job durations**: Approximately 80% of jobs show a significant mismatch between their requested and actual runtimes, with one-third terminated due to underestimation. This discrepancy results in poor scheduling decisions, including inflated queue times and inefficient resource allocation, fundamentally undermining the effectiveness of current schedulers.
- **Highly periodic communication patterns in multi-GPU jobs**: Multi-GPU workloads demonstrated recurring communication patterns, including predictable bursts of NVLink traffic and synchronization phases. These periodic behaviors present opportunities for system designers to develop predictive resource allocation strategies, minimizing interference and optimizing the scheduling of tasks.

These findings show great opportunities to improve GPU cluster efficiency through dynamic resource sharing and intelligent scheduling. Using insights from fine-grained monitoring data, system designers can create multitasking solutions that enhance GPU utilization while ensuring performance isolation. Such advancements could transform the management and utilization of research GPU clusters, making them more effective and cost-efficient.

2 Monitoring Framework and Environment

We have developed and deployed a fine-grained monitoring tool to analyze activities on a university GPU cluster.

2.1 GPU Cluster Overview

We conducted our study on a university research cluster comprising 89 nodes interconnected through a shared general-purpose networked file system [Schmuck and Haskin 2002]. The cluster is shared by researchers across multiple departments and includes the following hardware configurations:

- High-Performance Nodes (20): Each equipped with AMD EPYC Rome CPUs (128 cores), 768 GB RAM, and two NVIDIA A100 PCIe 40GB GPUs.
- Standard Nodes (69): Each featuring Intel Ice Lake CPUs (48 cores), 100 GB RAM, and four NVIDIA A100 SXM4 80GB GPUs.

All nodes are equipped with a 100 Gb/s ConnectX-6 InfiniBand NIC and a 1 Gb/s Ethernet NIC for out-of-band traffic.

2.2 Design Goals and Challenges

To effectively analyze GPU utilization without disrupting cluster operations, we established the following key design goals for the monitoring system:

- Minimal Overhead: The monitoring tool must have minimal impact on cluster performance, with low CPU, memory, and storage usage, as data collection was planned over several months across hundreds of GPUs.
- Operational Simplicity: The tool needed to be straightforward to deploy and operate, requiring minimal configuration and maintenance to avoid burdening system administrators.
- Fine-Grained Metrics: Capturing high-resolution metrics for Streaming Multiprocessors (SMs) utilization, memory utilization, SM occupancy, PCIe traffic, and NVLink traffic, Tensor Core Usage, providing a comprehensive view of GPU performance.
- Robust and scalable: The tool should be resilient to node failures and can be deployed to a large-scale GPU cluster.

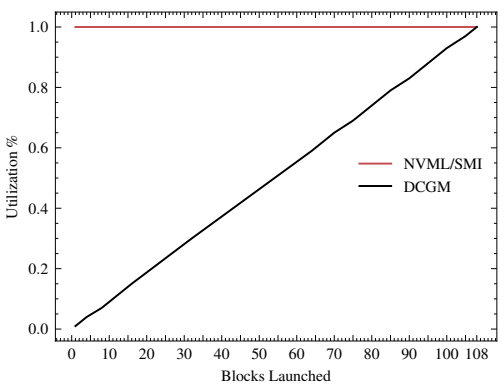
We have observed that the standard `nvidia-smi` metric, commonly used to measure GPU utilization, often misrepresents actual resource usage. It reports only the fraction of cycles during which a kernel is running or a copy engine is active, failing to account for fractional SM usage. For example, a job using just one SM out of 108 on an A100 GPU could still be reported as having 100% utilization.

To address this limitation, we utilized the "SM Activity" metric from Nvidia's DCGM API [Nvidia 2020c], which provides a more precise measure of GPU compute utilization. This fine-grained metric allowed us to observe intra-GPU spatial utilization and uncover significant discrepancies in reported utilization.

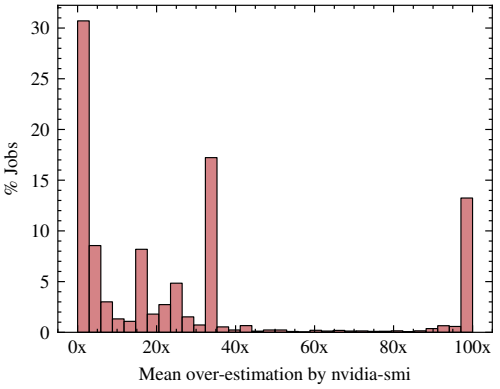
Figure 1a illustrates the discrepancy between `nvidia-smi` reported utilization and actual GPU usage by showing an example of an infinite-loop kernel (20 seconds) launched with a variable number of blocks (on an A100 with 108 SMs). DCGM-reported utilization gives an accurate view of intra-GPU spatial utilization. After deploying the monitoring system we compared the real-world differences. Figure 1b highlights the ratio of average utilization reported by the two approaches. This systematic discrepancy has significant implications for cluster efficiency analysis and scheduling decisions, especially in scenarios involving multitasking.

2.3 Design, Implementation and Deployment

To meet the design requirements, we design a monitoring tool to run on each host node, recording its data to a database. The main loop of the tool queries DCGM along with `/proc` to retrieve metrics of each attached GPU at a given frequency, write the returned results to a buffer, and then go to sleep. When the buffer is full, the tool writes its content to its dedicated DB. At a much lower



(a) Running an infinitely looping kernel on a variable number of SMs. The popularly used NVML/SMI metric may significantly over-estimate the actual fraction of GPU SMs used.



(b) Distribution of per-job utilization overestimates reported by `nvidia-smi` compared to the DCGM metric, stratified by job duration.

Fig. 1. Comparisons of nvidia-smi and actual (reported by DCGM) GPU SM utilization.

frequency (e.g. every hour), the tool queries the GPU cluster job scheduler (currently SLURM) for job information and writes to a separate table in the DB.

To achieve the low-overhead and efficiency goals, we develop the monitoring tool as standalone executable, written in a few hundred lines of C. We setup a SQLite DB for each host node to avoid any synchronization among multiple hosts. This design decision makes the monitoring tool robust (immune to host failures), though it requires some work to aggregate the DB data offline for data analysis.

For our data collection, the sample period is configured to 100ms, aligning with modern OS preemptive scheduling time slices. The local buffer size is configured to 600 samples to amortize the overhead of writing to DB. With such configurations, the buffer consumes < 2 MB. On a host node with 4 GPUs, the sample loop takes about 800 microseconds. At each 60 seconds, the tool spends about 60ms to write $\approx$ 24k rows totaling 1 MB to DB. Thus, the overhead is $\approx 0.9\%$ on a single CPU core. The total DB storage is about 1.5GB/day per node. The overhead on GPUs is negligible. Because of its minimal overhead and robustness, the university IT administrators allow us to deploy the monitoring system on a production research GPU cluster.

Our tool is open source and available at: Anonymized

3 Job Characteristics

Job Distribution by Department

During the 10-day period, 7,399 jobs were submitted by 150 users from 12 departments. Table 1 summarizes the distribution of jobs by department (inferred from users' group affiliations) and the proportion of GPU time reserved by each department, providing insights into the cluster's diverse usage patterns. Computationally intensive fields such as Chemistry and Chemical & Biological Engineering dominate GPU usage, while departments like Neuroscience and Sociology, though contributing fewer jobs, add to the cluster's workload diversity.

Job Duration and Categorization

Department	# of Jobs	Prop. of Reserved GPU-Time
Astronomy	136	5.6%
Chem & Bio Eng.	2184	22.7%
Chemistry	2837	38.2%
Comp. Science	306	4.9%
Elect. Eng.	227	4.6%
Genomics	119	0.9%
Mech. Eng	241	11.6%
Neuroscience	385	3.4%
Op. Research	181	0.7%
Physics	569	3.1%
Psychology	28	1.0%
Sociology	90	1.6%

Table 1. Breakdown of jobs in dataset by department. Departments which reserved ≤ 0.5% of overall GPU-time are omitted.

Figure 2 presents the cumulative distribution function (CDF) of job elapsed time, which we categorized into four groups for easier analysis:

- Short: <600 seconds (10 minutes)
- Medium: 600–7200 seconds (10 minutes–2 hours)
- Long: 7200–28800 seconds (2–8 hours)
- Very Long: >28800 seconds (8 hours)

Each category represents roughly 25% of the jobs. The CDF reveals discrete jumps, suggesting that jobs are often submitted with specific common time limits, possibly reflecting users' reliance on predefined configurations.

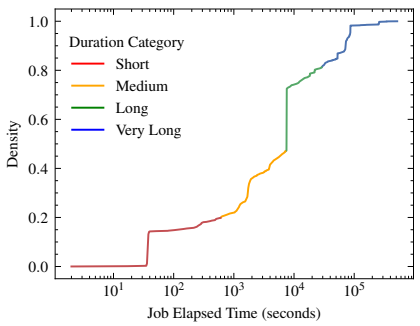


Fig. 2. CDF of job elapsed time (N=7399). Colored segments correspond to our different duration categories.

GPU Resource Requests

Figure 3a illustrates the distribution of jobs based on the number of GPUs and nodes requested at submission. A significant majority (95%) of jobs request a single GPU, with only a small fraction using multiple GPUs. However, multi-GPU and multi-node jobs take a disproportionately large amount of the GPU time on the cluster. These findings indicate that most workloads have limited parallelization in our dataset, potentially exacerbating the under-utilization of I/O and network bandwidth. This trend could be attributed to the complexity of parallelizing workloads across multiple GPUs or the longer queue times associated with multi-GPU jobs. This hypothesis is supported by Figure 3b; Jobs which request multi-GPU or multi-node configurations have significantly longer queue times on average, potentially encouraging frustrated users to prefer issuing single-GPU jobs instead.

The predominance of single-GPU jobs suggests that it is a main factor causing the underutilization of the GPU cluster as well as its network and I/O resources. The high timeout rate underscores a

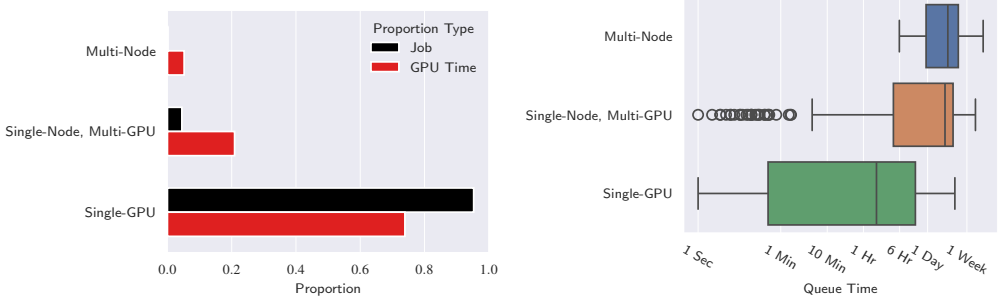
limitation with the job-scheduling paradigm, which requires users to pay good attention to their resource requests.

The predominance of single-GPU jobs appears to be a factor contributing to the underutilization of the GPU cluster, including its network and I/O resources. The high timeout rate highlights a limitation in the current job-scheduling paradigm, requiring users to carefully estimate and request appropriate resources for their jobs.

Completion State	Percentage of Jobs
Completed	60.70
Timeout	34.17
Failed	3.15
Canceled	1.99

Table 2. Completion states of jobs (N=7,399) on the Cluster. A significant portion (34.17%) of jobs timeout due to insufficient requested time. A smaller fraction are canceled by the user, or fail due to errors or host

4 Our Findings



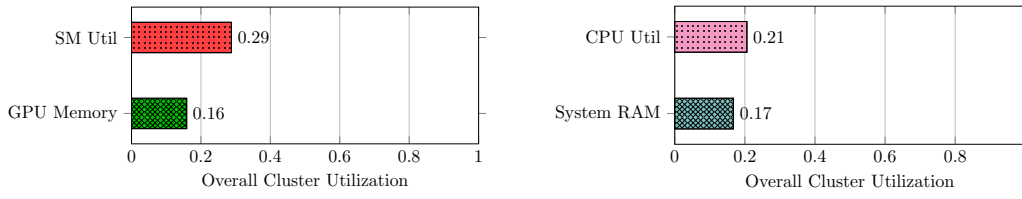
(a) (b)

Fig. 3. Classifying by Requested Resources. (*a*) Proportions of all jobs and total allocated GPU-Time and (*b*) Corresponding time spent in queue. We find multi-node and multi-gpu jobs face longer queue times than single-GPU jobs. Moreover, peak queue times even for single GPU jobs can exceed several days.

Significant Queuing. To better understand the cluster dynamics we categorize jobs into three groups: single-GPU, single-node with multi-GPUs, and multi-node. Figure 3b portrays the distributions of queue-times for each class; it is clear that requesting more resources results in longer queuing. The gap between requesting 1 GPU vs. multiple is significant. The median queue time for Single-GPU jobs was already nearly 2 hours. However, 2-hours does not seem long compared to the 2-*day* median queue time for multi-GPU requests. Given these dynamics, it makes sense that the vast majority of jobs only request a single-GPU.

Under-utilization. If jobs are sitting in a queue for so long, we might expect high-overall cluster utilization. Figure 4 displays the aggregate cluster utilization over this 10-day window. Despite the inflow of jobs, cluster-utilization remains low with an overall GPU compute utilization of 29% and GPU-memory utilization of 16%, with CPU resources also sitting largely unused. These aggregate statistics echo prior monitoring studies (see §5).

We investigate factors contributing to the widespread under-utilization and examine opportunities for future system designs.



(a) GPU Metrics (b) CPU Metrics

Fig. 4. Aggregate cluster utilization. Averaged over the entire cluster throughout the 10-day sample. Overall, we find both GPU and CPU resources to be under-utilized across the cluster.

4.1 Significant Mismatch between Requested vs. Actual Durations

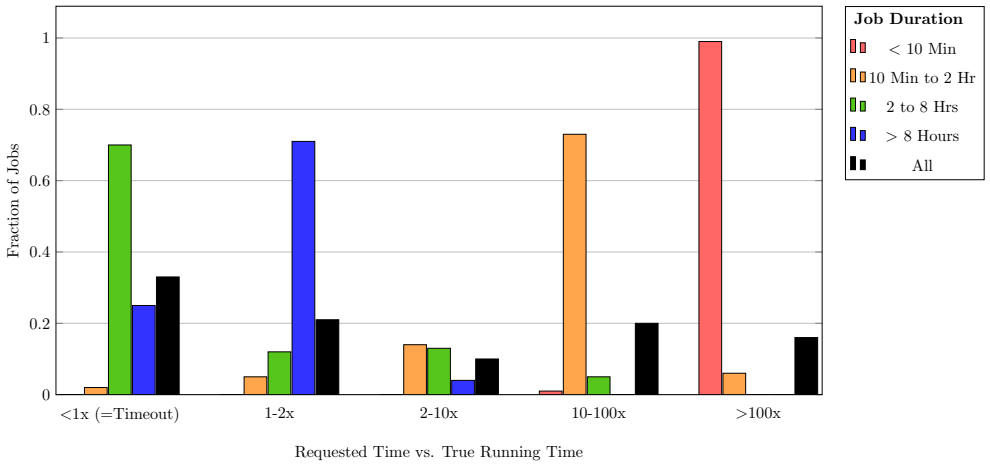


Fig. 5. A significant portion of jobs over-request time. This is particularly troublesome for the longer jobs which present difficulties for scheduling algorithms as they are receiving inaccurate inputs. This can result in poor global decisions and thus excessive queuing.

We find there is a widespread mismatch between requested and actual time durations of the monitored jobs. Figure 5 illustrates the ratio of requested time to actual elapsed time for jobs, categorized by requested job durations. With about 80% of jobs either underestimated or overestimated by a large factor. Only about 20% of requested durations fall within a factor of two of the actual runtime, and a substantial percentage are off by more than an order of magnitude.

Table 2 shows the distribution of job completion states. While 60% of jobs complete successfully, a notable **32% timeout due to insufficient requested time**, and the remaining 8% fail due to user cancellations, errors, or host memory constraints. The high timeout rate highlights that many users underestimate their job runtimes, leading to wasted GPU resources for these prematurely terminated jobs.

Over-requesting resources also has adverse effects. From the users' perspective, overestimating job durations increases queue delays, as the scheduler must allocate larger time slots for these jobs.

An important set of jobs are essentially idle during their actual execution time periods. Table 3 depicts the breakdown of jobs with fully-idle GPU-usage by duration. The vast majority, 1,043 jobs,

lasted less than a minute; of these 1,041 report successful completion, likely implying reservations for very small jobs or configuration errors where a GPU was accidentally requested.

Duration	Prop. of All Jobs	Prop. of Reserved GPU-Time
< 1 Minute	14.1%	0.02%
[1 Minute, 10 Minutes)	0.8%	0.001%
[10 Minutes, 2 Hours]	2.2%	0.44%
[2 Hours, 8 Hours]	0.4%	0.12%
> 8 Hours	0.6%	2.15%

Table 3. Fully-Idle GPU Jobs

A non-negligible number of jobs lasting multiple hours with fully-idle GPUs present a ripe opportunity for system designers. Out of all of the reserved GPU-time, 2.82% was spent in fully idle jobs, with 2.15% allocated as part of long-running (> 8 hours), completely idle jobs. Whether it be user-error in accidental GPU requests, or optimistic reservations that ended up not being needed, it is clear that an intelligent cluster would detect and take advantage of idle resources, rather than locking them away.

Implications

The widespread of drastic mismatch between requested and actual durations leads to job scheduler's poor scheduling decisions, causing low job completion rates, reduced cluster utilization, and increased queuing delays.

There are several significant implications:

- *Motivation for Avoiding User-Specified Time*: Our data reveal a severe and widespread mismatch between user-specified and actual durations, with one-third of jobs terminated due to underestimation. This discrepancy results in substantial waste of GPU resources and valuable human effort.
- *Motivation for Automatic Runtime Predictions*: Since the effectiveness of schedulers depends on the accuracy of their input data, using historical data or learning-based approaches could help schedulers make better decisions.
- *Motivation for Multitasking*: Multitasking approaches can significantly enhance GPU utilization by running multiple tasks concurrently, effectively leveraging underutilized resources.

These implications motivate systems designers to explore new metods to manage GPU clusters.

4.2 Great Opportunities for Multitasking

Most cluster managements require users to reserve entire GPUs for fixed durations, effectively "locking away" resources that often remain underutilized. While some jobs fully saturate their GPUs, our data shows that this is typically only for brief periods, if at all. Many reservations result in significant underutilization, raising the question of whether allocating entire GPUs on an all-or-nothing basis is the most effective approach, especially given the rising cost and demand for modern GPUs.

Low SM and Memory Utilizations

To study GPU utilization, we analyze the average Streaming Multiprocessor (SM) utilization and peak memory usage for single-GPU jobs, categorized by job duration. The results, illustrated in Figure 6, show significant underutilization:

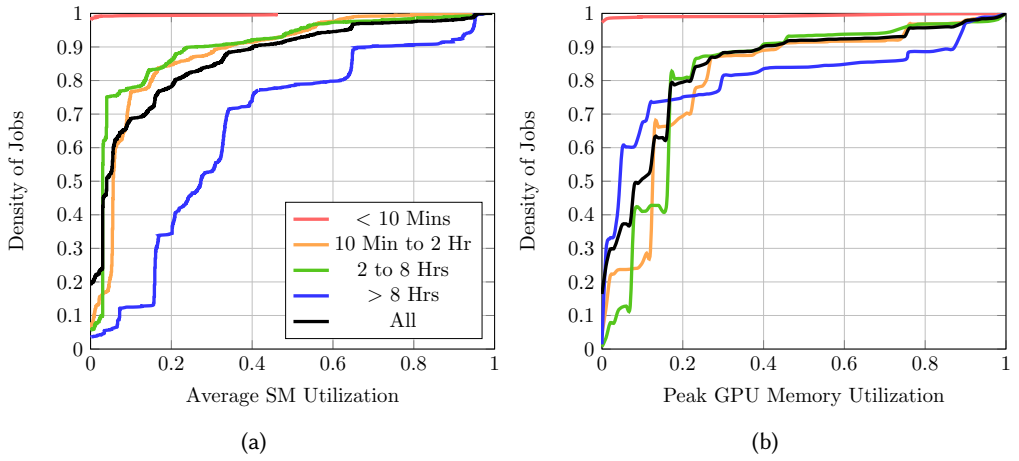


Fig. 6. CDFs of Single-GPU Job Utilization Metrics, stratified by duration. (a) Average GPU Utilization (b) Peak Memory Usage

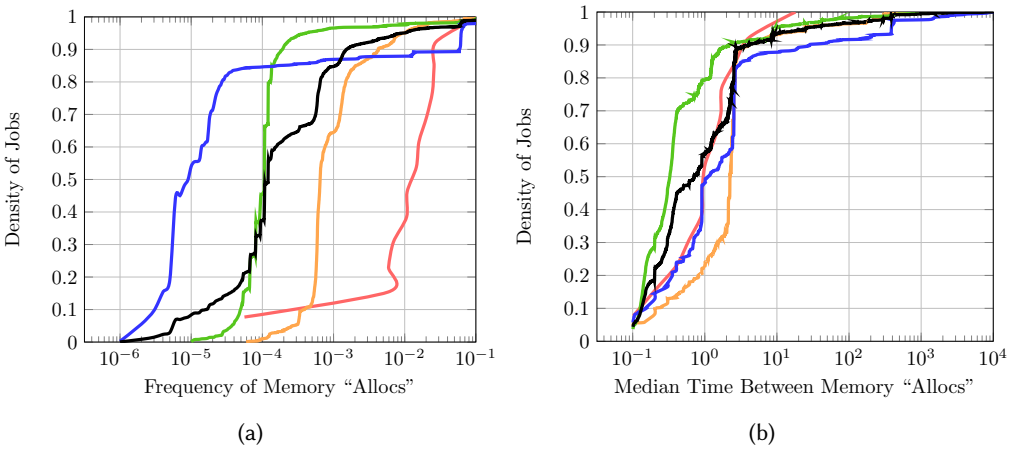
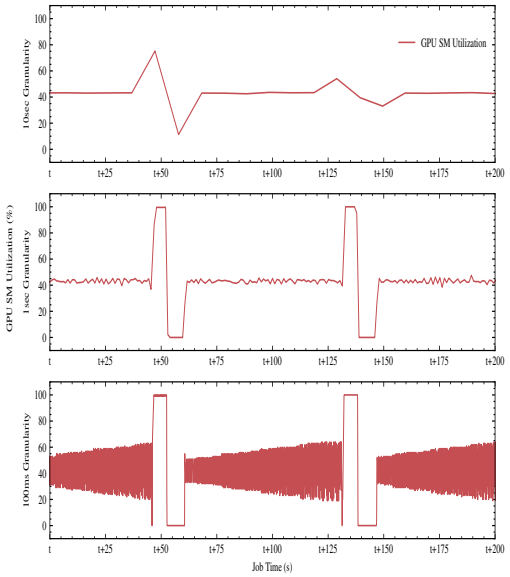


Fig. 7. Memory Stability: (a) Fraction of recorded samples where memory usage increased across all jobs (b) Median time between samples where memory usage increase, across all jobs.

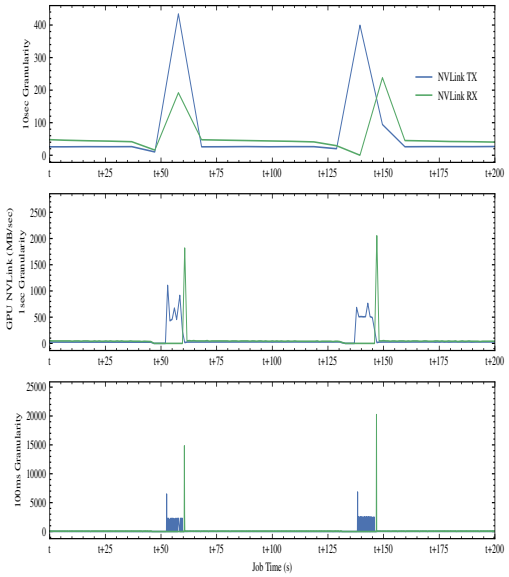
- 50% of jobs sustain an average SM utilization of under 5%, and
- 75% of jobs have a peak memory usage of less than 20% and an average SM usage of under 20%.

These observations suggest that most jobs are far from fully utilizing GPU capabilities, a trend likely to worsen as per-GPU performance grows faster than typical workload demands.

Efficient memory sharing is often a critical constraint for multitasking, as one task running out of memory can disrupt other tasks. Figure 6b shows that only 5% of jobs ever reach 75% of GPU memory capacity, and among long-running jobs (over 8 hours), just 10% exceed 90%. These results suggest that while some jobs may have large working sets, most leave ample memory headroom, making multitasking feasible.



(a) Comparison of querying GPU SM utilization at different sample rates. We see significant differences in observed program behavior as a result of the sampling rate; Details such as the high frequency fluctuations and deep troughs in SM utilization are lost when sampling at lower rates.



(b) Comparison of querying NVLink throughput at different sample rates. Note the differing y-axis. As granularity increases, we gain more insight into the burstiness of NVLink traffic.

Fig. 8. Comparisons of querying GPU SM utilization and NVLink throughput at different sample rates.

Figure 7 provides additional insights into memory stability. Memory-volatility data (Figure 7a) reveals how frequently memory usage increases during a job, while Figure 7b shows the typical time gap between memory growth events. Many jobs exhibit relatively stable memory usage for extended periods, reducing the risk of memory churn in multitasking scenarios.

Since single-GPU jobs are not gang-scheduled, they are well-suited for multitasking. We have performed a separate analysis of their SM and memory utilization, as illustrated in Figure 6a. The figure highlights the following:

- Only 10% of single-GPU jobs achieve an average SM utilization exceeding 50%.
- Among long-running jobs (over 8 hours), 33% surpass this threshold, indicating slightly higher but still limited utilization.
- Over 70% of single-GPU jobs utilize less than 20% of memory capacity.

These findings align closely with the overall utilization trends, as most jobs in the workload are single-GPU jobs.

Fine Temporal Granularity

The temporal granularity of GPU utilization sampling plays a crucial role in identifying underutilized intervals. Figure 8 compares sampling rates of 10 seconds, 1 second, and 100 milliseconds for both SM usage (Figure 8a) and NVLink throughput (Figure 8b). Coarser sampling masks significant fluctuations and bursty behavior, potentially misleading schedulers into overestimating GPU load. In contrast, 100ms sampling, as shown in the zoomed-in trace in Figure 9, captures the more accurate

peaks and troughs in SM utilization. Similarly, NVLink traffic reveals highly bursty patterns at fine-grained timescales.

These data demonstrate the importance of granular telemetry when considering optimal resource allocation. A scheduler can take advantage of fine-grained utilization data to dynamically enqueue additional tasks onto a GPU without waiting for the current job to complete.

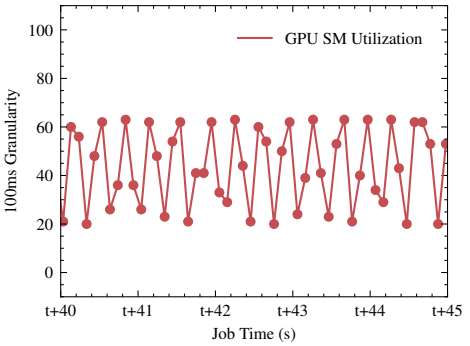


Fig. 9. Zoomed-in view of Figure 8a

Implications

Our data expose the severe underutilization of GPU's SMs and memory or great opportunities for multitasking on GPUs. There are several implications:

- *Motivation for multitasking*: Dynamic multitasking could unlock significant available resources, particularly 70% of jobs consume less than 20% SMs and memory.
- *Opportunity to Pack Single-GPU Jobs*: Single-GPU jobs are good candidates for multiplexing as they are not gang-scheduled and do not incur inter-GPU traffic. These jobs are account for a sustainable fraction of aggregate cluster usage.
- *Motivation for Using Fine-Grained Time Slices*: Fine-grained preemptive scheduling time slides such as 100-200ms, similar to the modern OS for CPUs, give more opportunities to improve GPU utilizations.

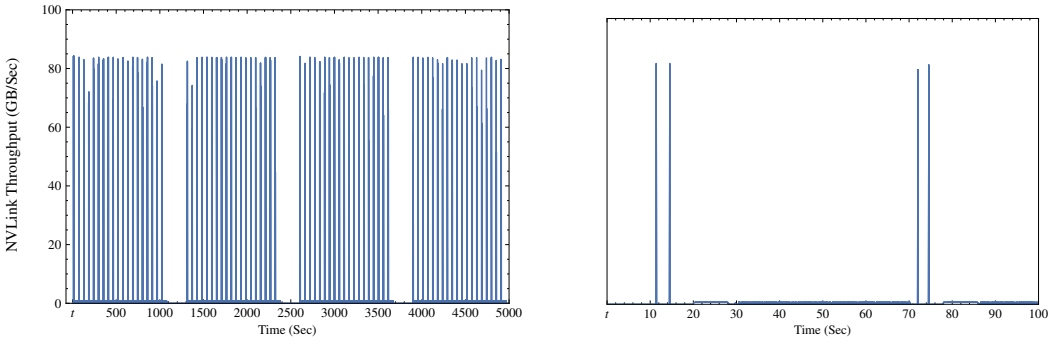
Nvidia's Multi-Instance GPU (MIG) [Nvidia 2020a], Multi-Process Service (MPS) [Nvidia 2020b] and Uniform Virtual Memory (UVM) [Nvidia 2017] are advancing toward greater maturity. These mechanisms offer promising opportunities to explore multitasking and unlock nearly 80% of underutilized GPU resources.

4.3 Highly Periodic, and Predictable, Communication Patterns

While most of the jobs on this cluster are isolated single-GPU jobs, figure 3a indicates that 26% of allocated GPU time was for parallel workloads, and presume this quantity to be significantly higher for AI and HPC dedicated clusters. The nature of parallel workloads, & accompanying synchronization, poses a significant challenge to multitasking. Unintended interference can have compounding straggler effects [Wu et al. 2024].

To determine the possibility of interleaving tasks alongside parallel workloads we investigated communication patterns of Multi-GPU workloads at a link utilization granularity. Given the large number of jobs, we present three representative examples.

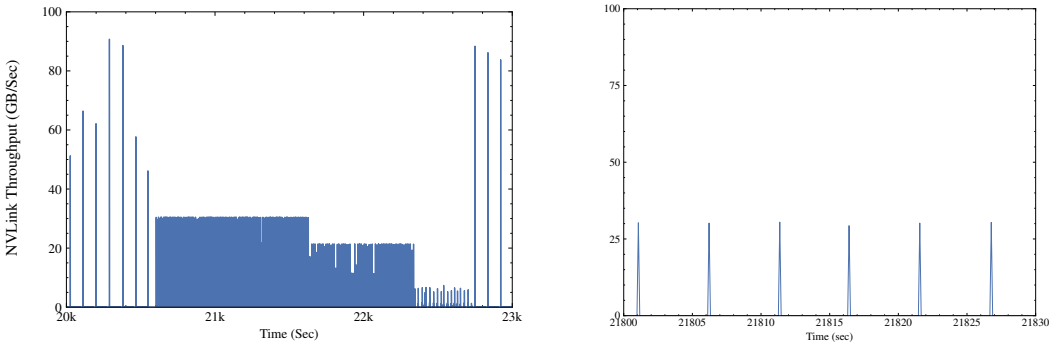
Single-Node Patterns



(a) ≈ 17-Minute Phase Periodicity

(b) Localized, Per-Minute, Bursts

Fig. 10. Single-Node Neuroscience Job



(a) "Bursty" Traffic at Two Different Rates

(b) Focusing in on denser rectangular regions in center of 11a still reveals sparsity and periodicity – once per 10 seconds vs. once per 60 seconds (surrounding spikes in 11a)

Fig. 11. Single-Node Psychology Job

4.3.1 Neuroscience Job. Figures 10a and 10b illustrate NVLink usage in a parallel neuroscience workload using 4 GPUs within a single node. This job demonstrates a periodic communication phase of approximately 17 minutes, with additional bursts occurring roughly once per minute. The left panel (Figure 10a) shows the consistent repetition of large-scale patterns throughout the job's runtime, while the right panel (Figure 10b) zooms in on the finer-grained bursts. This structured periodicity indicates that higher-level software systems can estimate NVLink traffic given a small initial sample of metrics.

4.3.2 Psychology Job. Figures 11a and 11b depict a psychology workload, also using 4 GPUs within a single node. This job exhibits mixed burst patterns, with bursts occurring at intervals of approximately 10 seconds and 60 seconds, respectively. While these intervals differ from the neuroscience example, the repetitive nature of NVLink traffic remains consistent, reflecting the structured communication demands of the application.

Comprehensive Picture

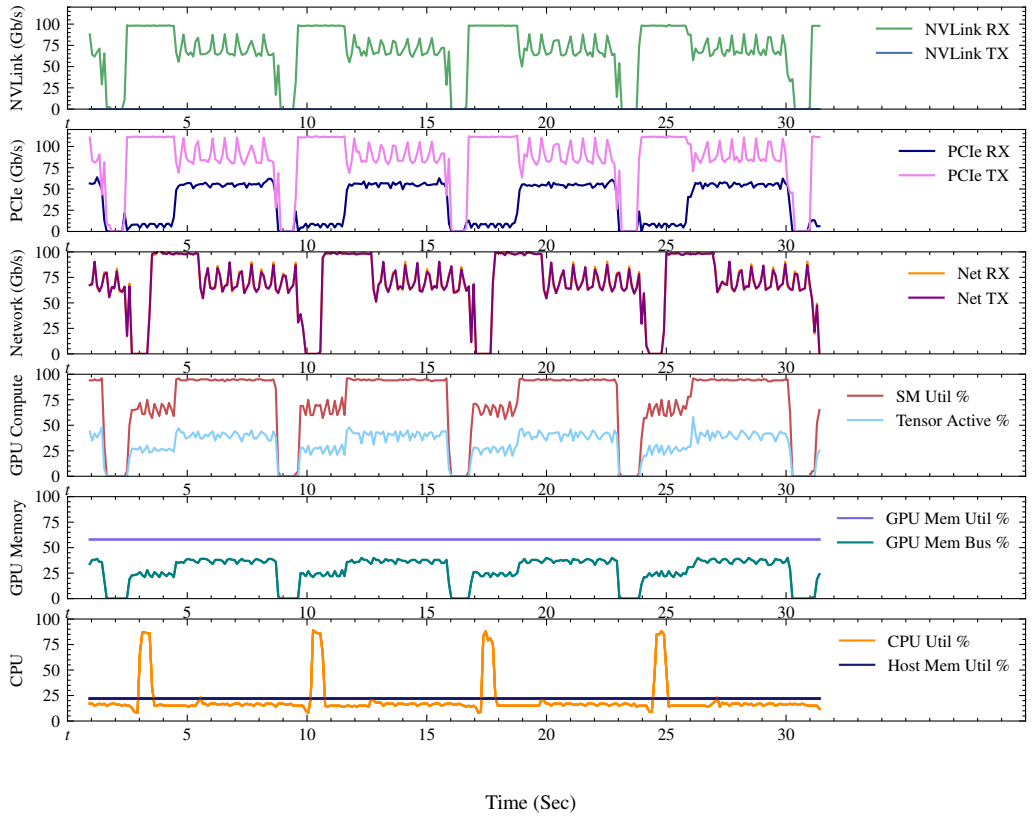


Fig. 12. Slice from 4-Node, 16-GPU workload: Periodicity across all resources & frequent pockets of SM idleness (4th row, red)

Periodicity among all resource dimensions can be seen in figure 12, captured from a 4-Node, 16-GPU EE job, likely doing AI training; this pattern repeats for 12 hours with almost no noise. We can infer communication behavior through a combination of traffic metrics. The network metrics come the node's singular InfiniBand NIC, while the interconnect metrics are for a specific GPU within the node. The PCIe TX/RX metric traffic includes bytes sent to/received from network card during RDMA in addition to bytes sent to/received from system memory. Note that interconnect graphs are in network units (Gb/s).

We see that this job is clearly bottlenecked by network bandwidth (purple/orange – the TX/RX lines are nearly identical), which reaches full saturation of ≈ 100Gb/s for 2-second windows. In this scenario, both NVLink and PCIe are constrained to the network limit.

We can deduce the selected GPU in 12 is continually transmitting over the network as the PCIe TX readings (in pink, second plot) align with the node's network TX readings. It is not transmitting to other intra-node GPUs (0 NVLink TX traffic from this selected GPU). The PCIe RX throughput implies inbound network traffic is split among multiple GPUs in this system. Other GPUs initially receive traffic (flat blue PCIe RX line) and then this GPU receives a partial share. These readings agree with the intra-node NVLink RX traffic this GPU receives (top plot in green) as it first is

reading at the network limit and then drops off when it is receiving some data directly from the NIC via RDMA.

We highlight that GPU SM Utilization (in red) fluctuates in lock-step with the overall communication pattern. Fully-idle GPUs result from what we presume to be global synchronization. In this example for every 7-second period, 1-second is spent idle. Accounting for 16-GPUs and a 12-hour duration, this implies around 21 hours of available GPU-computation that can be packed into these troughs.

The motif of small pockets of SM idleness, occurring at frequent and regular intervals, is the common case for many network-bound distributed GPU applications . Despite it's highly parallel nature, this type of job is a valid candidate to multi-task alongside. GPU memory usage is shown staying flat at 50% (it does not change for hours) in lavender (second from bottom), leaving ample space for other jobs' data.

Implications

Our analysis underscores the importance of high GPU-interconnect and network bandwidth. Single-Node jobs can exhibit full NVLink saturation during blocking communication phases. Figure 8 highlights how insufficient interconnect performance can degrade throughput during communication-intensive phases. Distributed jobs can be bottlenecked by network bandwidth and fail to saturate high-BW NVLinks and ultimately cause SM idle periods. The analysis indicates:

- *Opportunity for Accurate Traffic Predictions*: Repetitive communication patters present an opportunity for accurate traffic prediction. Scheduling tasks in an "out-of-phase" manner could help minimize contention and maintain consistent throughput.
- *Opportunity to Address Synchronization-Induced Idle Periods*: Transparent multitasking alongside long-running distributed jobs can significantly improve cluster utilization, especially in bandwidth-limited environments where communication delays often cause extended GPU idleness.

The general challenge to take advantage of these opportunities is to minimize interference during the bursty communication periods.

5 Related Work

Prior Work	Spatial Granularity	Temporal Granularity	Studied Workloads
Philly [Jeon et al. 2019]	Coarse	100 seconds	Deep Learning
AntMan [Xiao et al. 2020]	Coarse	$\approx$ 1 second[1]	Deep Learning
Helios [Hu et al. 2021]	Coarse	100 seconds	Deep Learning
Weng et al. [2022]	Coarse	15 seconds	Deep Learning
Moneo [Jiang et al. 2022]	Fine	Variable[2]	3 Individual DL Training Jobs[3]
Ours	Fine	100 ms	Mixed (12 disciplines)

Table 4. Comparison to several prior works studying cluster utilization. Prior works only use coarse-grained GPU utilization monitoring (NVML), and only consider time-scales up to 1-second granularity. Our work considers both fine-grained monitoring of SM utilization (DCGM) and sub-second temporal granularity.

GPU Monitoring APIs. Despite growing awareness of underutilization, few production systems have deployed sub-second sampling to characterize bursty or partial GPU usage. We summarize

key differences between our work and prior monitoring work in Table 4. The DCGM API supports high-resolution telemetry (e.g., SM Active, SM Occupancy, NVLink, PCIe) at 100–200 ms intervals, but it is rarely used at scale. To our knowledge, our work is among the first to deploy DCGM cluster-wide for an extended period and for a variety of workloads beyond large-scale deep learning. This approach reveals patterns —such as frequent GPU idleness, partial SM occupancy, and bursty interconnect usage—that remain hidden when sampling at intervals of one second or longer.

GPU Cluster Monitoring. Monitoring GPU-based clusters has been a significant topic of research, with numerous tools and frameworks designed to track resource usage. One of the earliest and most broadly adopted systems is *Ganglia* [Massie et al. 2004], which provides cluster-wide resource utilization metrics via a hierarchical design. Although Ganglia can be extended to expose GPU statistics (e.g., through an NVIDIA GPU monitoring module), it typically leverages the NVIDIA Management Library (NVML) [Nvidia 2007], offering only coarse-grained GPU usage data. Consequently, Ganglia-based monitoring lacks sub-second resolution and may miss rapid fluctuations in GPU compute, memory, or interconnect utilization. We instead designed our own monitoring solution to have more precise control over monitoring and sampling parameters.

We are not the first to champion fine-grained, cluster-wide GPU monitoring; Moneo [Jiang et al. 2022] is a tool developed by Microsoft which also adopts fine-grained monitoring of GPU metrics. While Jiang et al. propose capturing fine-grained metrics, they only study a limited selection of hand-picked deep learning training jobs. However, despite the availability of such tools, they remain unpopular for use on HPC research clusters; A recent study by Weakley et al. [2025] surveys GPU monitoring practices in HPC clusters, finding that the newer NVIDIA Data Center GPU Manager (DCGM) API [Nvidia 2020c]—which supports sub-second sampling—remains under-adopted. They attribute this to limited compatibility with older GPUs and the overhead of integrating DCGM with existing monitoring systems. We hope that our work serves to raise broader awareness.

In practice, the predominant monitoring infrastructure consists of deploying a container-based solution composed of Nvidia's `dcgm_exporter` (for GPU metrics) & the Prometheus monitoring framework [Rabenstein and Volz 2015] (for CPU-side metrics and & a time-series DB). This setup is targeted for Kubernetes-based clusters and is non-trivial to setup and control. Our monitoring tool can be deployed on any baremetal Nvidia Data-Center GPU server (without dependencies) and is trivial to start/stop from the command line (start/kill the daemon process).

Prior Characterization Studies. Several large-scale studies have examined GPU cluster utilization to identify inefficiencies and guide improvements in resource management; We provide a high-level summary of the differences with our work in Table 4. *Philly* [Jeon et al. 2019] from Microsoft was among the first to report considerable underutilization in a large-scale multi-tenant GPU cluster dedicated mostly to deep learning. Philly relied on NVML-based sampling at 100-second intervals, limiting its ability to capture sub-second bursts or fractional GPU usage. While underutilization in Philly was largely attributed to network constraints and gang scheduling, our results demonstrate that, even on nodes running a single job, significant idle periods occur that only become apparent when sampling at finer granularity. *Helios* [Hu et al. 2021] took a closer look at multiple production clusters running various GPU workloads. Like Philly, Helios employed NVML for coarse-grained (100-second) sampling and did not measure fractional SM utilization. Although Helios confirmed the prevalence of low GPU utilization and long queue times, it left open questions about the root causes of underutilization within each job's runtime. More recently, Weng et al. [2022] characterize utilization in a production cluster which supports time-multiplexed sharing of GPUs between jobs, demonstrating low contention for SM resources in their analysis. While a strong preliminary step towards improving utilization, our results show that significant fractions of GPU SM resources remain un-utilized by jobs; As time-slicing alone cannot allow for the sharing of these resources,

we feel fine-grained monitoring of utilization reveal greater opportunities for multiplexing. Our sub-second analysis highlights internal idle phases and scheduling opportunities that standard job-level metrics cannot readily detect.

GPU Multitasking and Scheduling Approaches. Many researchers have explored the idea of workload-independent GPU sharing [Adriaens et al. 2012; Chen et al. 2017; Liang et al. 2015; Mahajan et al. 2020; Park et al. 2015; Sedighi et al. 2024; Tanasic et al. 2014; Wang et al. 2016]. Much of recent work regarding GPU-multitasking has concentrated on machine-learning focused workloads. *Gandiva* [Xiao et al. 2018] considers time-slicing shared GPU resources between multiple jobs. *AntMan* [Xiao et al. 2020] introduces a software layer to multiplex GPU compute and memory resources, improving overall cluster usage. Like prior work, AntMan's monitoring relies on NVML at roughly 1 s intervals, thus it still may miss quickly shifting utilization patterns. Nevertheless, its reported improvements underline the potential of GPU multi-tenancy—a conclusion our results reinforce. By measuring SM usage at sub-second granularity, we demonstrate that jobs often leave enough unused SM capacity for other processes to co-locate effectively.

More recently, *Orion* [Strati et al. 2024] proposes a method for fine-grained sharing of GPU workloads on a single node, showing that performant and interference-aware multiplexing of GPU resources is possible. However, Orion relies on GPU utilization profiles collected offline, limiting its applicability to general user applications. We believe there is room for systems based on dynamic, real-time profiling of workloads to allow for flexible, application-agnostic sharing of GPUs. Based on the Helios dataset, Hu et al. [2021] proposed a "Quasi-Shortest-Service-First (QSSF)" scheduler, which uses historical job records to predict runtime and reduce queuing for large multi-GPU jobs. However, QSSF assumes relative stability in job behavior and easy access to training data, assumptions that may not hold in research environments where development and debugging are common. Our findings suggest that more adaptive scheduling strategies, fed by fine-grained performance metrics, could mitigate these shortcomings.

6 Limitations

Sampling Rate. We used the highest sampling frequency, 10 Hz, offered by DCGM [Nvidia 2020c]; however, we would have preferred to collect data at even higher rates if the API had allowed it, given our low-overhead monitoring tool supports rates 10–100× faster. While our temporal resolution exceeds that of prior monitoring studies, interconnect and network measurements are still aggregated over 100 ms windows, which may smooth out short-lived communication spikes and obscure burstiness. Moreover, individual GPU kernels often execute on a timescale of microseconds to single-digit milliseconds. Profiling tools such as Nvidia Nsight Systems (`nsys`) can capture this data (as a background process), but are not applicable for cluster-scale monitoring due to job degradation.

Duration of Study. Although we ran the monitoring tool for several months, we only retrieved precise mappings of Job ID to GPU ID(s) for the final 10 days. Rather than making potentially inaccurate inferences about which jobs ran on which devices, we limited our analysis to the clean, fully instrumented dataset.

Job Types and Diversity. Because most jobs in the cluster requested only a single GPU, we collected fewer samples of multi-GPU workloads (343 in total) and of these only 7 multi-node workloads. Our insight into the frequencies and behaviors of domain-specific workloads (e.g. ML or HPC workloads) is limited; our analysis aggregated from all departments.

Cluster Hardware and Heterogeneity. The cluster included two GPU types (A100 PCIe 40GB and A100 SXM4 80GB). We combined them into a single dataset for simplicity, as we report utilization

in percentages. However, these GPUs differ in their interconnect bandwidth (PCIe vs. NVLink). Because most jobs were single-node, the impact of these differences on our overall findings was small. Nonetheless, future work should investigate how heterogeneous hardware and network capabilities affect cluster utilization and communication patterns.

7 Conclusion

This paper presents a fine-grained analysis of a university research GPU cluster through a lightweight monitoring tool operating at 100ms granularity. Our study reveals three key findings: (1) severe underutilization, with over 70% of jobs using less than 20% of GPU memory and 50% of jobs averaging below 5% SM utilization; (2) pervasive mismatch between requested and actual job durations, affecting 80% of jobs and fundamentally limiting scheduler effectiveness; and (3) highly periodic communication patterns in multi-GPU jobs, suggesting opportunities for predictive resource allocation and compatible multiplexing.

These results, coupled with emerging technologies like MIG, MPS, and UVM, indicate significant potential for improving cluster efficiency through dynamic resource sharing and intelligent scheduling. By leveraging fine-grained monitoring data, system designers can develop multitasking solutions that maximize GPU utilization while maintaining performance isolation, potentially transforming how research GPU clusters are managed and utilized.

Acknowledgments

We thank Josko Plazonic, Jon Halverson, Curtis Hillegas, and the rest of the Princeton Research Computing team for helping deploy the monitoring tool and for insightful feedback on cluster environment. We also thank Sanjeev Arora, Danqi Chen, Karthik Narasimhan, and other members of the Princeton Language & Intelligence group for cooperation in our monitoring study.

References

Jacob T. Adriaens, Katherine Compton, Nam Sung Kim, and Michael J. Schulte. 2012. The case for GPGPU spatial multitasking. In *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture* (USA, 2012-02-25) *(HPCA '12)*. IEEE Computer Society, 1–12. doi:10.1109/HPCA.2012.6168946

Guoyang Chen, Yue Zhao, Xipeng Shen, and Huiyang Zhou. 2017. EffiSha: A Software Framework for Enabling Effficient Preemptive Scheduling of GPU. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, NY, USA, 2017-01-26) *(PPoPP '17)*. Association for Computing Machinery, 3–16. doi:10.1145/3018743.3018748

Fernando J. Corbató, Marjorie Merwin-Daggett, and Robert C. Daley. 1962. An experimental time-sharing system. In *Proceedings of the May 1-3, 1962, Spring Joint Computer Conference* (San Francisco, California) *(AIEE-IRE '62 (Spring))*. Association for Computing Machinery, New York, NY, USA, 335–344. doi:10.1145/1460833.1460871

F. J. Corbató, J. H. Saltzer, and C. T. Clingen. 1971. Multics: the first seven years. In *Proceedings of the May 16-18, 1972, spring joint computer conference (AFIPS '72 (Spring))*. Association for Computing Machinery, New York, NY, USA, 571–583. doi:10.1145/1478873.1478950

CJ Haddad. 2024. Nvidia CEO Jensen Huang says demand for next-generation Blackwell AI chip is 'insane'. https://www.cnbc.com/2024/10/03/nvidia-ceo-demand-for-blackwell-ai-chip-is-insane.html

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 (SC '21)*. Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3458817.3476223

Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, unjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '19)*. USENIX Association, USA, 947–960.

Yuting Jiang, Yifan Xiong, Lei Qu, Cheng Luo Luo, Chen Tian, Peng Cheng, and Yongqiang Xiong. 2022. Moneo: Monitoring Fine-grained Metrics Nonintrusively in AI Infrastructure. *ACM SIGOPS Operating Systems Review* 56, 1 (June 2022), 18–25. doi:10.1145/3544497.3544501

Yun Liang, Huynh Phung Huynh, Kyle Rupnow, Rick Siow Mong Goh, and Deming Chen. 2015. Efficient GPU Spatial-Temporal Multitasking. 26, 3 (2015), 748–760. doi:10.1109/TPDS.2014.2313342

Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and Efficient {GPU} Cluster Scheduling. 289–304. https://www.usenix.org/conference/nsdi20/presentation/mahajan

Matthew L Massie, Brent N Chun, and David E Culler. 2004. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Comput.* 30, 7 (July 2004), 817–840. doi:10.1016/j.parco.2004.04.001

Nvidia. 2007. NVIDIA Management Library (NVML). https://developer.nvidia.com/management-library-nvml

Nvidia. 2017. Unified Memory for CUDA Beginners. https://developer.nvidia.com/blog/unified-memory-cuda-beginners/

Nvidia. 2020a. Multi-Instance GPU User Guide. https://docs.nvidia.com/datacenter/tesla/pdf/NVIDIA_MIG_User_Guide.pdf

Nvidia. 2020b. Multi-Process Service User Guide. https://docs.nvidia.com/deploy/mps/index.html

Nvidia. 2020c. Welcome — NVIDIA DCGM Documentation latest documentation. https://docs.nvidia.com/datacenter/dcgm/latest/index.html

John K. Ousterhout. 1982. Scheduling techniques for concurrent systems. In *Proceedings of the 3rd International Conference on Distributed Computing Systems*. IEEE, 22–30.

Jason Jong Kyu Park, Yongjun Park, and Scott Mahlke. 2015. Chimera: Collaborative Preemption for Multitasking on a Shared GPU. 50, 4 (2015), 593–606. doi:10.1145/2775054.2694346

Björn Rabenstein and Julius Volz. 2015. Prometheus: A Next-Generation Monitoring System (Talk). USENIX Association, Dublin.

Frank Schmuck and Roger Haskin. 2002. GPFS, a Shared-disk File System For Large Computing Clusters. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies* (Monterey, California, 2002-01).

Hoda Sedighi, Daniel Gehberger, Amin Ebrahimzadeh, Fetahi Wuhib, and Roch H. Glitho. 2024. Efficient Dynamic Resource Management for Spatial Multitasking GPUs. (2024), 1–18. doi:10.1109/TCC.2024.3511548 Conference Name: IEEE Transactions on Cloud Computing.

Foteini Strati, Xianzhe Ma, and Ana Klimovic. 2024. Orion: Interference-aware, Fine-grained GPU Sharing for ML Applications. In *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 1075–1092. doi:10.1145/3627703.3629578

Ivan Tanasic, Isaac Gelado, Javier Cabezas, Alex Ramirez, Nacho Navarro, and Mateo Valero. 2014. Enabling preemptive multiprogramming on GPUs. 42, 3 (2014), 193–204. doi:10.1145/2678373.2665702

Zhenning Wang, Jun Yang, Rami Melhem, Bruce Childers, Youtao Zhang, and Minyi Guo. 2016. Simultaneous Multikernel GPU: Multi-tasking throughput processors via fine-grained sharing. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2016-03). 358–369. doi:10.1109/HPCA.2016.7446078 ISSN: 2378-203X.

Le Mai Weakley, Scott Michael, Laura Huber, Abhinav Thota, Ben Fulton, and Matthew Kusz. 2025. Monitoring and Characterizing GPU Usage. *Concurrency and Computation: Practice and Experience* 37, 3 (2025), e8341. doi:10.1002/cpe.8341 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.8341.

Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. {MLaaS} in the Wild: Workload Analysis and Scheduling in {Large-Scale} Heterogeneous {GPU} Clusters. 945–960. https://www.usenix.org/conference/nsdi22/presentation/weng

Tianyuan Wu, Wei Wang, Yinghao Yu, Siran Yang, Wenchao Wu, Qinkai Duan, Guodong Yang, Jiamang Wang, Lin Qu, and Liping Zhang. 2024. FALCON: Pinpointing and Mitigating Stragglers for Large-Scale Hybrid-Parallel Training. arXiv:2410.12588 [cs.DC] https://arxiv.org/abs/2410.12588

Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective Cluster Scheduling for Deep Learning. 595–610. https://www.usenix.org/conference/osdi18/presentation/xiao

Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. AntMan: dynamic scaling on GPU clusters for deep learning. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20)*. USENIX Association, USA, 533–548.

Chen Zhao, Wu Gao, Feiping Nie, and Huiyang Zhou. 2022. A Survey of GPU Multitasking Methods Supported by Hardware Architecture. *IEEE Transactions on Parallel and Distributed Systems* 33, 6 (2022), 1451–1463. doi:10.1109/TPDS.2021.3115630