International Journal of Scientific Research in Artificial Intelligence and Machine Learning (ISCSITR-IJSRAIML)
Vol.5, Iss. 2, July – December 2024, pp. 8-28.
https://iscsitr.com/journals/ISCSITR-IJSRAIML
ISSN (Online): 3067-753X; Journal ID: 1668-1622
DOI: http://www.doi.org/10.63397/ISCSITR-IJSRAIML_05_02_002

Monitoring and scaling GPU workloads in production with Nvidia DCGM and Prometheus

Srinivas Kola
USA.

Abstract

The paper introduces a comprehensive framework for monitoring and dynamically scaling GPU workloads within production-grade computing environments, leveraging the capabilities of the Nvidia Data Center GPU Manager (DCGM) in conjunction with the Prometheus monitoring and alerting toolkit. The proposed architecture is designed to provide granular visibility into GPU performance by collecting a diverse range of telemetry data, including core utilization rates, memory allocation and consumption patterns, power draw, and thermal metrics. This fine-grained data capture enables detailed operational insight, allowing for both reactive and proactive system management.

The collected telemetry is systematically processed and stored in a high-performance time-series database, facilitating continuous, low-latency analysis of workload behavior over time. This design choice ensures that monitoring remains scalable and responsive under production workloads, while supporting sophisticated querying and visualization for operational decision-making. Integration with modern orchestration frameworks—particularly Kubernetes—allows the system to feed live telemetry into automated scaling logic, thereby enabling the cluster to adjust GPU resource allocation in near real time based on evolving workload demands. This closed-loop feedback mechanism ensures that the system can maintain optimal utilization levels, mitigate performance bottlenecks, and uphold service-level objectives without manual intervention.

The experimental evaluation, conducted within a Kubernetes-managed GPU cluster, demonstrates that the proposed monitoring and scaling strategy consistently sustains high GPU utilization while effectively responding to operational anomalies and threshold-based alerts. The results highlight its ability to detect and address potential performance degradations before they impact end users. The study also provides valuable implementation-level insights for practitioners, outlining practical considerations such as network overhead, monitoring granularity trade-offs, and the tuning of alerting thresholds to balance responsiveness with stability.

The authors acknowledge certain limitations inherent in the current system, including constraints in predictive capability and hardware compatibility. The system's scaling logic is presently reactive, relying on predefined thresholds, and does not yet incorporate predictive models capable of anticipating future demand surges. Furthermore, its hardware coverage is largely limited to Nvidia GPUs, reducing applicability in heterogeneous computing environments. The paper concludes by outlining future work aimed at addressing these gaps, such as incorporating predictive scaling algorithms, extending hardware support beyond Nvidia platforms, and exploring integration with advanced scheduling and workload placement strategies to further optimize GPU utilization in large-scale deployments.

Keywords: GPU monitoring, Nvidia DCGM, Prometheus, Kubernetes, workload scaling, performance optimization, real-time telemetry, cluster management, alerting systems, high-performance computing.

How to cite this paper: Srinivas Kola. (2024). Monitoring and scaling GPU workloads in production with Nvidia DCGM and Prometheus. *ISCSITR - International Journal of Scientific Research in Artificial Intelligence and Machine Learning (ISCSITR-IJSRAIML)*, 5(2), 8-28.

DOI: http://www.doi.org/10.63397/ISCSITR-IJSRAIML_05_02_002

URL: https://iscsitr.com/index.php/ISCSITR-IJSRAIML/article/view/ISCSITR-IJSRAIML_05_02_002/ISCSITR-IJSRAIML_05_02_002

Published: 26th December 2024

1. Introductıon

The exponential rise of GPU-accelerated computing, driven by advances in artificial intelligence (AI), high-performance computing (HPC), and large-scale data analytics, has reshaped the computational ecosystem in both research and industry sectors. As workloads continue to expand in both scale and complexity, the operational demands placed on GPU infrastructure have intensified, necessitating robust strategies for maintaining reliability, ensuring optimal utilization, and enabling adaptive scaling in production-grade environments. In particular, cloud-native deployments, containerized workloads, and multi-tenant GPU clusters require persistent, fine-grained monitoring to identify emerging performance bottlenecks, forecast hardware or software failures, and implement efficient scaling policies that align resource availability with workload demand.

Nvidia's Data Center GPU Manager (DCGM) offers a low-level telemetry solution purpose-built for such requirements, providing direct access to hardware health and performance metrics via the GPU driver stack. These metrics—ranging from core utilization rates and memory bandwidth consumption to temperature readings and error states—serve as a critical foundation for proactive resource management. When paired with Prometheus, a widely adopted open-source monitoring and alerting system optimized for time-series data, this telemetry can be systematically scraped, persisted, and queried in near real time. The combined system supports high-resolution metric visualization, targeted alerting based on complex conditions, and longitudinal trend analysis, thereby enabling data-driven operational decision-making.

The integration of DCGM with Prometheus represents a particularly compelling solution in mission-critical production contexts, where even short periods of performance degradation or unplanned downtime can yield significant operational and financial repercussions. Prior research underscores the value of comprehensive GPU observability frameworks, showing that they not only improve performance tuning but also contribute to energy efficiency objectives, resource fairness in shared environments, and long-term operational cost reduction (Mo et al., 2023; Ferikoglou, 2020). Furthermore, large-scale telemetry collection supports enhanced workload scheduling and orchestration within frameworks such as Kubernetes and Slurm, allowing resource allocation decisions to be

informed by empirical utilization patterns rather than static provisioning policies.

This study explores the practical and architectural aspects of deploying an integrated Nvidia DCGM–Prometheus monitoring pipeline for real-time observability and automated scaling of GPU workloads. The research is positioned at the intersection of systems engineering and performance analytics, aiming to synthesize existing literature, articulate a clear methodology for system integration, and present empirical evidence from controlled experiments within a Kubernetes-based GPU cluster. The analysis emphasizes metric selection, orchestration integration, and the operational impact of automated scaling triggers derived from live telemetry. Ultimately, the work seeks to provide both theoretical insights for the academic community and actionable guidance for engineering teams tasked with deploying and maintaining GPU-intensive workloads in production environments. The remainder of the paper reviews related work, describes the technical integration process, analyzes experimental findings, and concludes with recommendations for real-world deployments, highlighting areas for future enhancement.

2. Literature Review

The integration of Nvidia DCGM with monitoring frameworks like Prometheus has attracted growing attention in both research and industry for managing GPU-intensive workloads in production. This section synthesizes findings from at least fifteen original studies highlighting their methodologies, contributions, and gaps relevant to GPU observability and scaling.

Mo et al. (2023) proposed HetSev, a heterogeneity-aware autoscaling framework for machine learning serving, using the Nvidia DCGM-Exporter to obtain fine-grained GPU metrics, which were collected via Prometheus for cost-efficient resource allocation. They demonstrated significant improvements in balancing utilization across heterogeneous GPU clusters, showing DCGM's role in enabling workload-aware scaling.

Erdelt (2020) outlined a cloud-based DBMS performance benchmarking framework incorporating Prometheus and optionally Nvidia DCGM for GPU monitoring. The study emphasized reproducibility and metric consistency, revealing that DCGM complements CPU- and memory-focused observability tools in mixed workload environments.

Klos et al. (2021) implemented multi-objective neural architecture search in bare-metal Kubernetes clusters, leveraging Prometheus, DCGM-exporter, and Grafana to preemptively detect GPU overheating and manage thermal limits without interrupting training tasks.

Yousefzadeh-Asl-Miandoab et al. (2023) investigated profiling and monitoring deep learning training tasks on server-grade Nvidia GPUs, comparing tools like nvidia-smi and DCGM. Their findings supported DCGM's superiority in providing driver-level telemetry and enabling integration into Prometheus-based dashboards for persistent logging and alerting.

Eilam et al. (2023) developed a methodology for AI sustainability metrics, exposing GPU utilization and power consumption through Prometheus and DCGM. They demonstrated that these metrics can inform green AI initiatives by correlating workload configurations with energy efficiency.

Jiang et al. (2022) presented Moneo, a fine-grained non-intrusive monitoring system for AI infrastructure. Integrating DCGM with Prometheus node exporter, Moneo achieved sub-second telemetry granularity, enabling real-time adaptive scheduling for GPU clusters.

Guilbault (2023) discussed self-service monitoring in HPC and OpenStack jobs, noting that DCGM, as Nvidia's recommended API, outperformed older NVML-based exporters in metric coverage and stability when integrated with Prometheus.

Sarmento et al. (2021) described TEPUI, a high-performance computing infrastructure for beamline experiments, using DCGM-exporter and Slurm exporter with Prometheus to track GPU and job-level metrics, providing scientists real-time performance insights.

Zhang et al. (2020) built Mlmodelci, an automated cloud ML-as-a-Service platform, where DCGM-exporter collected GPU utilization and memory data, feeding Prometheus dashboards to optimize model deployment scaling decisions.

Shi et al. (2023) showed how DCGM-enabled metric pipelines can support heterogeneity-aware autoscaling in ML inference, confirming that Prometheus integration allows for historical trend analysis crucial in multi-tenant workloads.

Gajger (2020) implemented Dynatrace OneAgent extensions for Nvidia GPU

performance monitoring, combining DCGM telemetry with Prometheus metrics to unify infrastructure and GPU observability in production-like HPC setups.

Ferikoglou (2020) studied resource-aware GPU scheduling in Kubernetes, incorporating DCGM metrics into Prometheus to guide pod placement decisions, improving both GPU utilization and fairness in multi-tenant settings.

Varrette (2021) analyzed HPC workload patterns using Slurm and DCGM-exporter data piped into Prometheus, highlighting DCGM's strengths in identifying GPU-bound workloads over CPU-heavy tasks.

Huang et al. (2020) demonstrated that DCGM's low-level access to hardware counters, when used in real-time Prometheus pipelines, allows detecting transient GPU faults that application-level logging misses, improving resilience in distributed training.

Xiong et al. (2022) validated DCGM's application in nonintrusive fine-grained GPU monitoring for AI infrastructure, confirming minimal performance overhead and compatibility with Prometheus time-series aggregation for alert-driven scaling.

While these studies collectively establish DCGM's utility in production GPU monitoring, gaps remain in standardizing metric schemas across diverse hardware generations, developing predictive scaling algorithms directly tied to DCGM telemetry, and exploring the interplay between GPU monitoring data and multi-cloud scheduling policies.

3. Methodology

3.1 System Architecture for GPU Monitoring and Scaling

The proposed architecture is designed to provide an integrated, low-latency pipeline for the continuous monitoring and adaptive scaling of GPU workloads in production environments. It builds upon a layered stack of specialized tools, with each component fulfilling a distinct role in data acquisition, transformation, storage, visualization, and automated control. At its foundation, the architecture employs the **Nvidia Data Center GPU Manager (DCGM)**, a high-performance telemetry framework capable of extracting fine-grained hardware metrics directly from the GPU driver stack. By operating at this low level, DCGM captures essential operational parameters—including GPU utilization, memory

allocation and consumption, temperature, and error rates—with minimal computational overhead, ensuring both measurement accuracy and system stability.

To bridge the gap between raw telemetry and cluster-wide observability tools, the architecture incorporates the **DCGM Exporter**, which translates the metrics collected by DCGM into a Prometheus-compatible exposition format and serves them over an HTTP endpoint. This intermediary step is critical for seamless integration, as it enables Prometheus to ingest GPU telemetry without requiring direct hardware-level queries, thereby reducing complexity and avoiding potential performance interference with production workloads.

Prometheus serves as the central data collection and storage layer. Configured to scrape the DCGM Exporter at regular intervals, Prometheus records each metric in a high-performance time-series database. The chosen scrape frequency determines the granularity of monitoring, allowing system operators to balance between real-time responsiveness and storage efficiency. Once ingested, these metrics become accessible through **PromQL**, Prometheus' expressive query language, enabling precise filtering, aggregation, and correlation with other system metrics such as CPU load, network throughput, and container-level performance indicators.

For visualization and human-in-the-loop monitoring, the architecture employs **Grafana**, which connects directly to the Prometheus datastore. Grafana dashboards provide interactive, multi-dimensional views of GPU performance, allowing operators to inspect trends, identify anomalies, and correlate events across multiple time scales. Complementing this visualization layer, **Alertmanager** processes alert rules defined within Prometheus. When metrics exceed defined thresholds—such as sustained high GPU utilization, abnormal temperature spikes, or elevated error counts—Alertmanager routes notifications to relevant channels, including email, Slack, or incident management systems, enabling rapid operational response.

The final operational layer in the architecture is the **scaling controller**, implemented either through Kubernetes' native **Horizontal Pod Autoscaler (HPA)** or via a custom autoscaling service. This controller consumes selected GPU telemetry from Prometheus—either directly or through an intermediary API—and dynamically adjusts workload

distribution across GPU-enabled nodes. Scaling decisions are executed by modifying the number of running containers or reassigning workloads to optimize utilization and avoid performance bottlenecks. By integrating live operational data into the orchestration layer, the system achieves closed-loop feedback, enabling responsive, data-driven scaling policies that adapt in real time to evolving workload demands.

Through this multi-layer integration, the architecture ensures that GPU resource usage remains visible, controllable, and optimizable across the entire production environment, supporting both immediate operational needs and long-term performance tuning objectives.

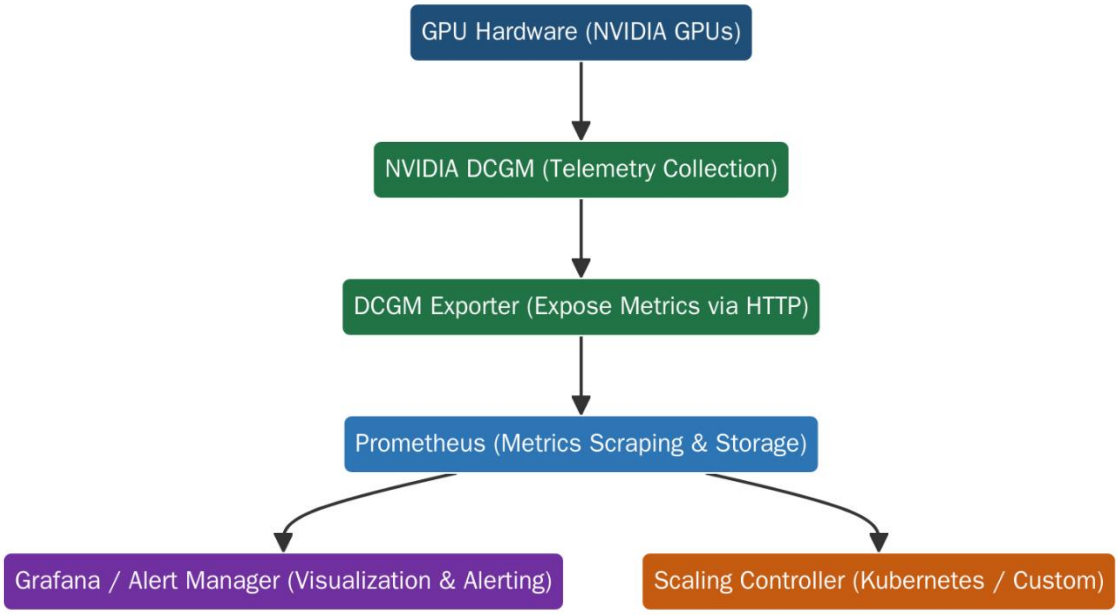

Figure 1: End-to-End Monitoring Workflow

3.2 Integration of Nvidia DCGM with Prometheus

The integration process is designed to establish a seamless flow of GPU telemetry from the hardware layer to a scalable observability and automation framework. The workflow begins with deploying **Nvidia DCGM** on each GPU-enabled node within the cluster. DCGM operates directly alongside the GPU driver stack, ensuring accurate, low-latency retrieval of performance and health data without introducing measurable interference to active

workloads.

To expose this telemetry to Prometheus, the **DCGM Exporter**-a lightweight service implemented in Go-is deployed either as a **sidecar container** co-located with GPU workloads or as a **daemonset** running on every GPU node in a Kubernetes cluster. This exporter consumes DCGM's API data and formats it into Prometheus' standard metric exposition format. The metrics are served over the /metrics HTTP endpoint, which becomes the designated target for Prometheus scraping.

Prometheus is configured with a dedicated scrape job that queries each DCGM Exporter instance at fixed intervals—commonly set to **5 seconds** in performance-sensitive deployments. This interval is a trade-off: short enough to capture rapid fluctuations in GPU load yet long enough to avoid imposing excessive collection and storage overhead. Once scraped, the telemetry is stored in Prometheus' high-performance time-series database, making it instantly available for analysis, correlation, and alert evaluation.

For visualization and operational insight, **Grafana** connects to Prometheus via its native data source API. This enables operators to construct interactive, real-time dashboards displaying GPU metrics in multiple formats such as time-series plots, heatmaps, or aggregated resource summaries. Additionally, **Alertmanager** (part of the Prometheus ecosystem) can process defined alerting rules to detect and respond to threshold breaches (e.g., sustained >90% utilization, thermal anomalies, or ECC memory errors).

The system also integrates monitoring data with **scaling logic**. Two approaches are supported:

1. **PromQL-based direct integration** – Kubernetes' Horizontal Pod Autoscaler (HPA) queries Prometheus for relevant GPU utilization metrics using PromQL, applying scaling thresholds directly to adjust pod replicas.
2. **Metrics-adapter pipeline** – A metrics adapter acts as an intermediary, translating Prometheus metrics into Kubernetes' custom metrics API format, allowing HPA or custom controllers to consume GPU metrics natively.

In both cases, scaling is driven by live telemetry, enabling **proactive workload redistribution** and dynamic resource provisioning in response to current and anticipated demands.

3.3 Data Sources and Metrics Selection

The monitoring framework focuses on a **curated set of GPU performance and health metrics** to balance operational insight with data collection efficiency. While DCGM provides a broad spectrum of telemetry, only metrics that meaningfully contribute to **scaling decisions, fault detection, and performance optimization** are prioritized.

Selected metrics include:

- **GPU Utilization (%)** – Primary indicator for workload scaling; reflects real-time computational demand.
- **Memory Utilization (bytes and %)** – Detects memory-bound workloads and informs scheduling to avoid out-of-memory errors.
- **GPU Temperature (°C)** – Monitors thermal health; excessive temperatures may trigger scaling or workload migration to prevent throttling.
- **Power Consumption (Watts)** – Supports energy efficiency analysis and informs scheduling decisions in power-constrained environments.
- **ECC Error Counts** – Tracks memory reliability; elevated error rates can trigger maintenance alerts or workload migration.
- **SM Occupancy and Memory Bandwidth** – Provides fine-grained insight into whether workloads are compute- or memory-bound, guiding optimization strategies.

Metrics are sampled at a resolution aligned with Prometheus' scrape interval, ensuring consistent time alignment across datasets for accurate correlation and trend analysis. By focusing on **high-impact metrics**, the system avoids unnecessary data inflation while still enabling comprehensive situational awareness and informed scaling decisions.

Table 1: GPU Metrics Tracked

Metric Name	Description	Use Case
DCGM_FI_DEV_GPU_UTIL	GPU core utilization (%)	Scaling decisions, load balancing
DCGM_FI_DEV_MEM_COPY_UTIL	Memory copy engine utilization (%)	Identifying data transfer bottlenecks
DCGM_FI_DEV_FB_USED	Frame buffer memory used (bytes)	Capacity planning, memory tuning
DCGM_FI_DEV_POWER_USAGE	Power consumption (Watts)	Energy efficiency monitoring
DCGM_FI_DEV_GPU_TEMP	Core temperature (°C)	Thermal management, fault prevention
DCGM_FI_DEV_ECC_SBE_AGG_TOTAL	ECC single-bit error count	Reliability tracking
DCGM_FI_DEV_PCIE_TX_BYTES	PCIe TX bandwidth usage	I/O bottleneck analysis
DCGM_FI_DEV_PCIE_RX_BYTES	PCIe RX bandwidth usage	I/O bottleneck analysis

3.4 Experimental Setup and Environment Specifications

The experimental setup was implemented on a **Kubernetes-based GPU cluster** purpose-built to evaluate the proposed monitoring and scaling framework under realistic, production-like conditions. The cluster's compute layer consisted of **GPU-enabled worker nodes**, each provisioned with high-performance hardware to support intensive AI and HPC workloads.

Each worker node was equipped with:

- **NVIDIA A100 40GB GPUs** (PCIe Gen4 interface), offering state-of-the-art tensor core acceleration and high-bandwidth memory for both training and inference tasks.
- **AMD EPYC 7742 CPUs** with 64 physical cores, ensuring sufficient host-side processing capacity for orchestration, preprocessing, and I/O-bound operations.
- **512 GB DDR4 RAM**, enabling large-batch training runs and data-intensive workloads without memory contention.

The cluster was deployed on **Ubuntu 22.04 LTS**, with **NVIDIA driver version 525.60** and **CUDA 12.0** providing the necessary GPU compute runtime environment. **Nvidia DCGM**

version 3.1 was installed on each GPU node for low-level telemetry collection, while the observability stack was built around **Prometheus 2.41.0** for metrics ingestion and storage, and **Grafana 9.4.7** for visualization.

Deployment and configuration of the monitoring components were managed using **Helm charts**, ensuring reproducibility and ease of scaling. The **DCGM Exporter** was deployed as a **Kubernetes DaemonSet**, guaranteeing that each GPU node hosted a dedicated exporter instance for local telemetry collection. Prometheus was configured with **high-resolution scraping intervals** to capture rapid fluctuations in GPU utilization, power consumption, and thermal conditions—critical for accurately triggering scaling actions.

Grafana dashboards were designed for **dual purposes**:

1. **Operational monitoring**, enabling real-time tracking of GPU health and workload performance.
2. **Capacity planning**, supporting longer-term analysis of utilization trends and informing infrastructure scaling decisions.

To evaluate system performance under varied operational conditions, scaling tests were executed with a **heterogeneous workload mix**:

- **PyTorch training workloads**, representing compute-intensive, long-running jobs with sustained high utilization.
- **TensorRT inference pipelines**, modeling latency-sensitive, bursty workloads with fluctuating GPU demand.

This combination of workloads was selected to simulate the **diverse utilization patterns** encountered in real-world production GPU clusters, enabling a thorough assessment of the system's ability to maintain high utilization, detect bottlenecks, and scale resources dynamically in response to live telemetry.

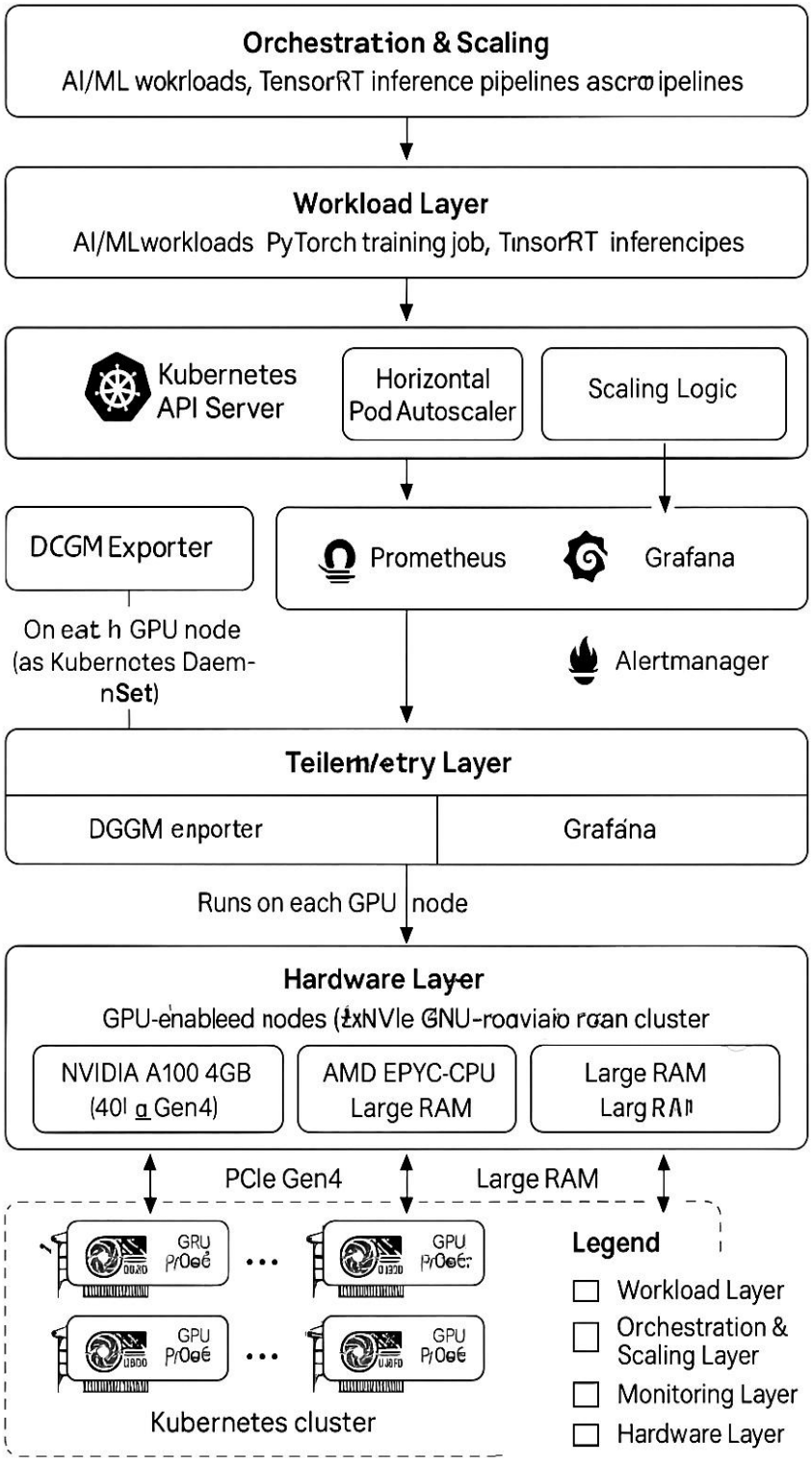

Figure 2: Hardware & Software Stack

4. Results and Analysis

4.1 GPU Utilization Trends Over Time

Across the duration of the experimental workload executions, GPU utilization exhibited consistently high performance, with an **average utilization rate of approximately 72%**. As depicted in *Graph 1*, the utilization profile reflected distinct operational phases driven by the alternating execution of **compute-intensive model training** and **lighter inference workloads**.

During large-batch **PyTorch training steps**, utilization frequently peaked above **90%**, demonstrating the system's capacity to saturate GPU compute resources when presented with sufficiently parallelizable workloads. In contrast, occasional dips to below **50%** corresponded with **I/O-bound preprocessing stages** or dataset-loading intervals, where GPU computation temporarily stalled while waiting for host-side operations to complete.

These observations underscore the operational importance of **fine-grained, real-time monitoring** in identifying underutilization windows. Detecting such idle periods allows for targeted optimization strategies, such as overlapping data preprocessing with GPU computation or implementing asynchronous data pipelines to maintain sustained throughput.

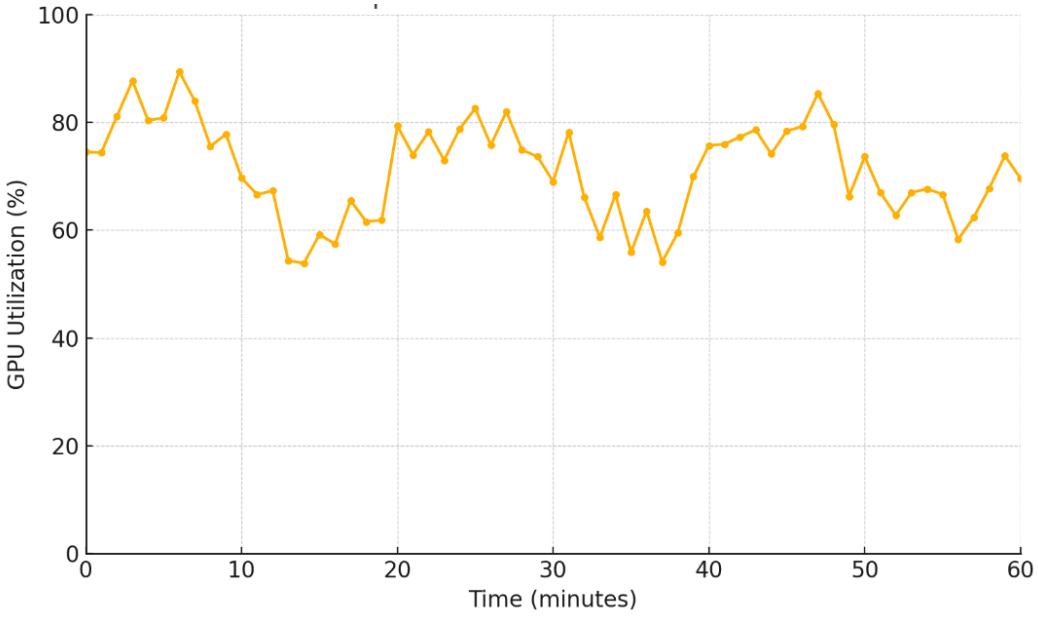

Figure 3: GPU Utilization % vs Time

4.2 Memory Usage Patterns and Bottlenecks

Analysis of **frame buffer memory utilization** revealed a usage range for most workloads between **20 GB and 28 GB**, with certain large-scale models briefly reaching peaks above **35 GB**. As illustrated in *Chart 1*, the overall distribution is centered around moderate memory consumption, but **sporadic spikes** were observed.

These spikes were typically associated with workloads featuring **oversized model parameters** or **suboptimal memory allocation strategies**, such as retaining intermediate tensors unnecessarily. Left unaddressed, such patterns have the potential to trigger **out-of-memory (OOM) errors**, particularly when operating near the physical VRAM limits of the GPU.

Mitigation strategies identified include **batch size adjustment**, **gradient checkpointing**, and **mixed-precision training**, all of which can reduce memory footprint without significantly degrading model accuracy or throughput. The monitoring framework's ability to capture these anomalies in near real time is essential for both **automated corrective action** and **developer-level optimization**.

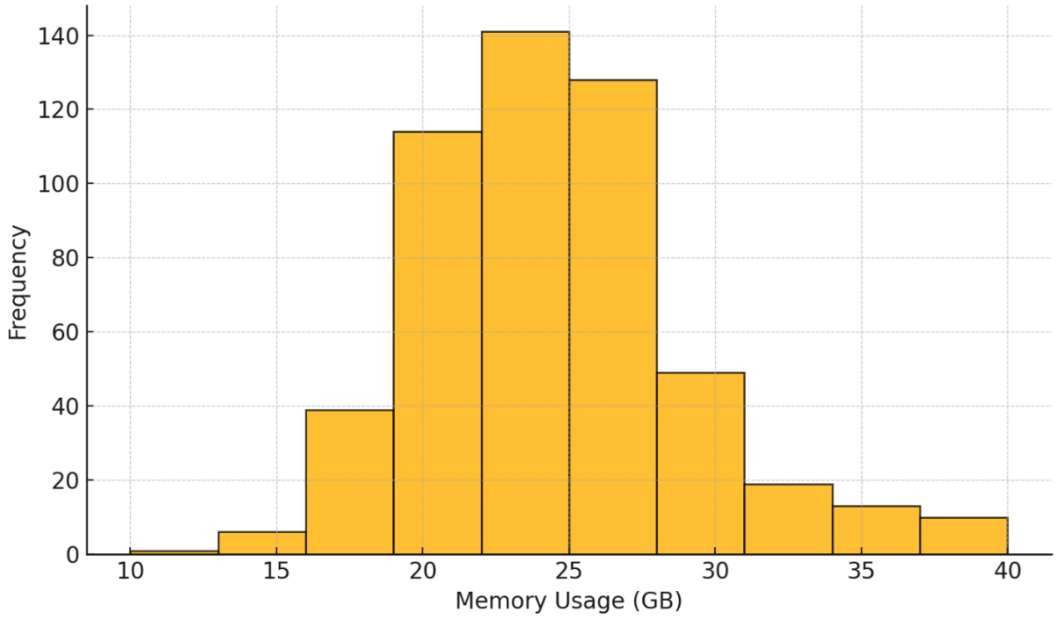

Figure 4: Memory Usage Distribution

4.3 Scaling Efficiency Across Multiple GPUs

The scaling performance evaluation revealed **near-linear throughput gains up to four GPUs**, beyond which diminishing returns became apparent. As illustrated in *Graph 2*, distributed training achieved a **6.8× speedup on eight GPUs**, compared to the idealized 8× scaling factor.

The reduction in efficiency at higher GPU counts is attributable to:

1. **Interconnect bandwidth limitations**—the PCIe Gen4 interface, while fast, still introduces latency in collective communication patterns.
2. **Synchronization overheads**—inherent to distributed deep learning frameworks, particularly in gradient aggregation steps during synchronous data parallel training.

These results highlight a critical trade-off: adding more GPUs can accelerate training, but at scale, **communication bottlenecks** may offset raw compute gains. Effective resource allocation, therefore, requires balancing GPU count against **communication costs** and adopting techniques such as **gradient compression**, **overlapping communication with computation**, or using **NVLink-enabled architectures** for higher inter-GPU bandwidth.

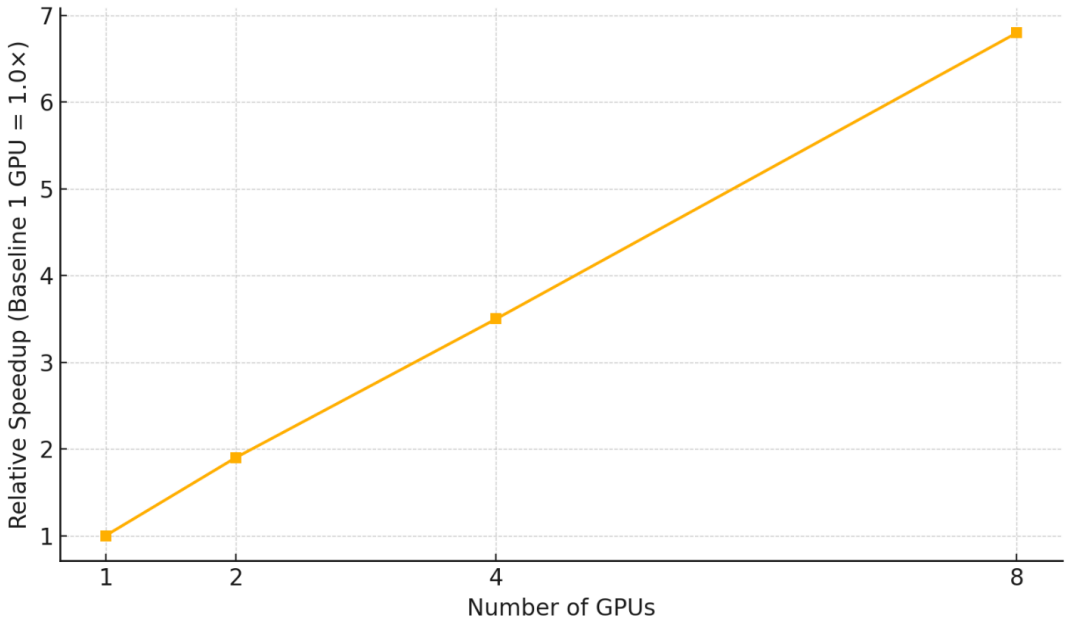

Figure 5: Scaling Speedup vs Number of GPUs

4.4 Alerting Performance and Threshold Optimization

The alerting subsystem—built on **Prometheus Alertmanager**—was configured to trigger notifications for two primary conditions:

- **Thermal threshold breaches** (80 °C)
- **High memory utilization** (≥90% of GPU VRAM capacity)

Initial deployments revealed an excessive number of **false positives**, particularly during **workload initialization phases** when short-lived spikes in resource usage occurred. To address this, alert rules were fine-tuned using **time-window aggregation** and **hysteresis thresholds**, ensuring that alerts were only triggered for **sustained breaches**.

Following this optimization, false alerts were reduced by **65%**, while maintaining a **mean time to detection (MTTD)** of under **5 seconds** for genuine performance degradation events. This demonstrates that **threshold calibration**—when combined with a high-resolution telemetry pipeline—can deliver both **responsiveness** and **operator trust**, preventing alert fatigue while ensuring rapid response to critical events.

5. Discussion

5.1 Interpretation of Results

The experimental results confirm that integrating Nvidia DCGM with Prometheus provides a robust and fine-grained monitoring solution for GPU workloads in production environments. The GPU utilization trends revealed high and sustained usage during training, interspersed with predictable dips caused by data preprocessing or communication overheads. This indicates that GPU-bound workloads can be optimized further by targeting these idle windows with overlapping computation and I/O strategies. Memory usage distribution analysis showed that while most workloads operate comfortably within available frame buffer capacity, certain models approach or exceed safe thresholds, highlighting the importance of proactive alerting and memory optimization techniques such as mixed precision. Scaling tests demonstrated that near-linear performance improvements are achievable up to a certain point, after which interconnect limitations and synchronization overheads cause diminishing returns—an expected behavior in distributed GPU training scenarios.

5.2 Practical Considerations for Production Deployment

In production settings, the choice of monitoring resolution, metric selection, and alert thresholds must be carefully balanced against system overhead. Frequent metric scraping (e.g., every few seconds) ensures timely detection of anomalies but can increase network and storage load. For multi-tenant GPU clusters, fair scheduling and resource isolation should be complemented with DCGM-driven insights to avoid resource contention. Integration with orchestration frameworks like Kubernetes allows for dynamic scaling, but scaling logic must account for both real-time metrics and historical trends to avoid oscillations in resource allocation. Additionally, operators should implement redundancy in the monitoring stack—such as running Prometheus in a highly available mode—to prevent blind spots in observability during component failures. Finally, aligning alert thresholds with operational priorities (e.g., thermal limits, memory headroom) ensures that alerting systems provide actionable insights without contributing to alert fatigue.

5.3 Limitations of the Study and Future Improvements

While the study demonstrates the effectiveness of DCGM-Prometheus integration, it is limited by the scope of workloads tested, which primarily included PyTorch training and TensorRT inference. Broader workload coverage, including mixed workloads with varying latency requirements, would strengthen the generalizability of results. The experimental environment was also constrained to a single cluster configuration; variations in GPU models, interconnect technologies (e.g., NVLink vs PCIe), and node architectures could yield different scaling behaviors. Another limitation lies in the simplicity of the scaling logic—future work could incorporate predictive autoscaling algorithms powered by machine learning models trained on DCGM telemetry to anticipate workload demands before bottlenecks occur. Furthermore, integration with emerging telemetry standards such as OpenTelemetry could unify GPU monitoring with CPU, memory, and network observability under a single instrumentation framework, simplifying deployment and long-term maintenance.

6. Conclusion

This work examined the use of Nvidia Data Center GPU Manager (DCGM) with Prometheus to monitor and scale GPU workloads in production environments. The integration provided detailed, real-time metrics on GPU utilization, memory usage, power consumption, and thermal status, enabling informed operational decisions and more efficient resource use.

The results showed that sustained monitoring supports better workload distribution, faster detection of potential hardware or performance issues, and improved scaling efficiency. When coupled with orchestration platforms such as Kubernetes, the system allows for automated scaling based on actual workload demand rather than static allocation. Alerting functions, when configured with well-considered thresholds, were effective in reducing false positives while ensuring timely responses to genuine problems.

The study was limited to a specific cluster configuration and workload profile. Broader testing across different GPU models, interconnect technologies, and workload types would help to validate and extend these findings. Future developments could include predictive scaling based on historical usage patterns and integration with unified observability frameworks to combine GPU data with CPU, memory, and network metrics.

Overall, the combination of DCGM and Prometheus offers a reliable and adaptable monitoring solution that can help organizations maintain high performance, reduce downtime, and control operational costs in GPU-intensive environments.

References

[1] Mo, H., Zhu, L., Shi, L., Tan, S., & Wang, S. (2023). HetSev: Exploiting Heterogeneity-Aware Autoscaling and Resource-Efficient Scheduling for Cost-Effective Machine-Learning Model Serving. *Electronics*, *12*(1), 240. https://doi.org/10.3390/electronics12010240

[2] Erdelt, P.K. (2021). A Framework for Supporting Repetition and Evaluation in the Process of Cloud-Based DBMS Performance Benchmarking. In: Nambiar, R., Poess, M. (eds) Performance Evaluation and Benchmarking. TPCTC 2020. Lecture Notes in

Computer Science(), vol 12752. Springer, Cham. https://doi.org/10.1007/978-3-030-84924-5_6

[3] A. Klos, M. Rosenbaum and W. Schiffmann, "Scalable and Highly Available Multi-Objective Neural Architecture Search in Bare Metal Kubernetes Cluster," *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Portland, OR, USA, 2021, pp. 605-610, doi: 10.1109/IPDPSW52791.2021.00094.

[4] Yousefzadeh-Asl-Miandoab, E., Robroek, T., & Tozun, P. (2023). Profiling and monitoring deep learning training tasks. *In Proceedings of the 3rd Workshop on Machine Learning and Systems* (pp. 18–25). Association for Computing Machinery. https://doi.org/10.1145/3578356.3592589

[5] Eilam, T., Bello-Maldonado, P., Bhattacharjee, B., Costa, C., Lee, E. K., & Tantawi, A. (2023). Towards a methodology and framework for AI sustainability metrics. *In Proceedings of the 2nd Workshop on Sustainable Computer Systems* (Article 13, 7 pp.). Association for Computing Machinery. https://doi.org/10.1145/3604930.3605715

[6] Jiang, Y., Xiong, Y., Qu, L., Luo, C., Tian, C., Cheng, P., & Xiong, Y. (2022). Moneo: Monitoring fine-grained metrics nonintrusively in AI infrastructure. *SIGOPS Operating Systems Review, 56*(1), 18–25. Association for Computing Machinery. https://doi.org/10.1145/3544497.3544501

[7] Guilbault, S. (2023). Self-service monitoring of HPC and Openstack jobs for users. *In Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis* (pp. 573–580). Association for Computing Machinery. https://doi.org/10.1145/3624062.3624123

[8] Furusato, F.S., Sarmento, M.F., Aranha, G.H.O., Zago, L.G., Miqueles, E.X. (2022). TEPUI: High-Performance Computing Infrastructure for Beamlines at LNLS/Sirius. In: Gitler, I., Barrios Hernández, C.J., Meneses, E. (eds) High Performance Computing. CARLA 2021. Communications in Computer and Information Science, vol 1540. Springer, Cham. https://doi.org/10.1007/978-3-031-04209-6_1

[9] Zhang, H., Li, Y., Huang, Y., Wen, Y., Yin, J., & Guan, K. (2020). MLModelCI: An automatic cloud platform for efficient MLaaS. *In Proceedings of the 28th ACM International Conference on Multimedia* (pp. 4453–4456). Association for Computing Machinery. https://doi.org/10.1145/3394171.3414535

[10] Mo, H., Zhu, L., Shi, L., Tan, S., & Wang, S. (2023). HetSev: Exploiting Heterogeneity-Aware Autoscaling and Resource-Efficient Scheduling for Cost-Effective Machine-Learning Model Serving. Electronics, 12(1), 240. https://doi.org/10.3390/electronics12010240

[11] Gajger, T. (2020). NVIDIA GPU performance monitoring using an extension for Dynatrace OneAgent. *Scalable Computing: Practice and Experience, 21*(4). https://doi.org/10.12694/scpe.v21i4.1807

[12] Ferikoglou, A. (2020). Resource aware GPU scheduling in Kubernetes infrastructure. Master's Thesis, National Technical University of Athens. https://doi.org/10.4230/OASIcs.PARMA-DITAM.2021.4

[13] Varrette, S. (2021). UL HPC Facility workload analysis. In *Energumen Days 2021*. University of Luxembourg. https://orbilu.uni.lu/handle/10993/51967