Scholars Bulletin *(Engineering)*
An Official Publication of "Scholars Middle East Publishers" ISSN 2412-9771 (Print)
Dubai, United Arab Emirates ISSN 2412-897X (Online)
Website: www.saudijournals.com

Intelligent Gpu Scheduling and Fairness Mechanisms for Multi-Tenant Ai Workloads in Kubernetes–Openstack Environments

Lova Gautham Pemmadi[1*], Hema Sree Chunduri[2], Praveen Veeramachaneni[2]
[1]Southern Arkansas University
[2]Northwestern Polytechnic University

***Corresponding author**
Lova Gautham Pemmadi

Article History
Received: 14.11.2018
Accepted: 16.12.2018
Published: 30.12.2018

DOI:
10.36348/sb.2018.4.12.9

Abstract: The proliferation of artificial intelligence (AI) and deep learning workloads has intensified demand for Graphics Processing Unit (GPU) resources in cloud computing environments. Multi-tenant infrastructures, particularly those leveraging Kubernetes orchestration within OpenStack platforms, face critical challenges in efficiently sharing GPU resources while maintaining fairness across diverse tenants and workloads. This paper investigates intelligent GPU scheduling and fairness mechanisms tailored for multi-tenant AI workloads in Kubernetes–OpenStack environments. Building upon recent advances in container orchestration and GPU virtualization, this study examines the architectural integration of Kubernetes device plugins with OpenStack Nova and Ironic GPU management components. The analysis explores fairness versus performance trade-offs, evaluating how priority-based queuing, workload-aware preemption, and policy-driven scheduling impact training latency, inference throughput, and cost efficiency. Through comprehensive examination of existing GPU sharing techniques, virtualization approaches, and scheduling algorithms, this research identifies critical design considerations for achieving balanced resource allocation. The findings reveal that hybrid scheduling approaches combining time-slicing with spatial partitioning, coupled with adaptive fairness policies, offer superior performance isolation and tenant satisfaction compared to static allocation schemes. Furthermore, the integration of capacity-based resource models with dynamic workload profiling enables fine-grained quality-of-service (QoS) guarantees essential for latency-sensitive inference tasks while maximizing utilization for batch training workloads. This work contributes to the growing body of knowledge on GPU resource management in containerized cloud environments and provides practical insights for deploying fair and efficient multi-tenant AI infrastructures.

Keywords: GPU scheduling, fairness mechanisms, multi-tenancy, Kubernetes, OpenStack, container orchestration, AI workloads, resource allocation.

1. INTRODUCTION

The exponential growth of machine learning and deep neural network applications has positioned GPUs as indispensable computational resources in modern cloud infrastructures. Unlike traditional CPU-centric workloads, AI applications exhibit distinct characteristics including irregular memory access patterns, variable execution times, and heterogeneous resource requirements spanning training and inference phases (Becchi *et al.,* 2012). Multi-tenant cloud environments, where multiple users and organizations share physical infrastructure, must address the dual challenges of maximizing GPU utilization while ensuring equitable resource distribution. Container orchestration platforms, particularly Kubernetes, have emerged as dominant frameworks for deploying and managing distributed applications. When integrated with Infrastructure-as-a-Service (IaaS) platforms such as OpenStack, Kubernetes enables flexible, scalable deployment of containerized workloads with sophisticated resource management capabilities (Patchamatla, 2018). However, GPU resources present unique scheduling complexities absent in CPU or memory management. GPUs are discrete, non-preemptible devices with limited support for fine-grained sharing, making traditional fair-share scheduling algorithms inadequate (Hong *et al.,* 2017a).

The challenge intensifies in multi-tenant scenarios where diverse workloads, ranging from long-running training jobs to latency-critical inference services, compete for limited GPU capacity. Static allocation strategies, where entire GPUs are exclusively assigned to single containers or virtual machines, result in poor utilization as many AI workloads exhibit bursty GPU usage patterns (Goswami *et al.,* 2016). Conversely, naïve sharing approaches without proper isolation mechanisms lead to performance interference, unpredictable execution times, and tenant dissatisfaction (Sengupta *et al.,*

2013). Recent research has explored various GPU virtualization and scheduling techniques, including API remoting, para-virtualization, and hardware-assisted virtualization (Hong *et al.,* 2017b). Middleware solutions have demonstrated promising results in enabling fine-grained GPU sharing through kernel-level interception and time-slicing mechanisms (Goswami *et al.,* 2016). However, the integration of these techniques within production-grade orchestration platforms like Kubernetes, coupled with the specific architectural constraints of OpenStack deployments, remains an active area of investigation. This paper addresses the critical gap in understanding how intelligent scheduling policies and fairness mechanisms can be effectively implemented for multi-tenant AI workloads in Kubernetes–OpenStack environments. The research extends the foundational work of Patchamatla (2018) on optimizing Kubernetes-based multi-tenant container environments by specifically focusing on GPU-aware scheduling strategies, fairness guarantees, and performance trade-offs. The primary objectives include: (1) analyzing architectural integration points between Kubernetes device plugins and OpenStack GPU management, (2) evaluating fairness mechanisms and their impact on workload performance, (3) examining trade-offs between strict fairness and overall system efficiency, and (4) identifying design principles for policy-driven GPU schedulers in containerized AI infrastructures.

The remainder of this paper is organized as follows. Section 2 reviews related work on GPU virtualization, scheduling algorithms, and multi-tenant resource management. Section 3 describes the architectural components of Kubernetes–OpenStack GPU management. Section 4 analyzes fairness mechanisms and scheduling policies. Section 5 examines performance implications and trade-offs. Section 6 discusses implementation considerations and challenges. Section 7 concludes with future research directions.

2. Related Work

2.1 GPU Virtualization Techniques

GPU virtualization forms the foundation for enabling multi-tenant GPU sharing in cloud environments. Hong *et al*. (2017b) provided a comprehensive taxonomy of GPU virtualization approaches, categorizing techniques into API remoting, para-virtualization, and hardware-assisted methods. API remoting intercepts GPU library calls and forwards them to remote GPU servers, enabling location transparency but introducing communication overhead. Para-virtualization modifies guest operating systems to enable direct GPU access through hypervisor mediation, balancing performance and isolation (Gupta *et al.,* 2011). Kato *et al*. (2012) introduced Gdev, an operating system-level approach providing first-class GPU resource management through virtual memory abstraction and device memory sharing. Their work demonstrated that OS-level virtualization enables fine-grained isolation and improved data management compared to hypervisor-based approaches. Similarly, Becchi *et al*. (2012) developed a virtual memory-based runtime supporting GPU multi-tenancy across cluster nodes, achieving significant performance improvements over serialized execution through dynamic binding and load balancing.

2.2 GPU Scheduling Algorithms

Scheduling algorithms determine how GPU resources are allocated among competing workloads, directly impacting both fairness and efficiency. Sengupta *et al*. (2013) proposed Rain, a multi-level scheduler implementing system-level GPU "hyperthreading" with prioritization and least-attained-service fairness for server workloads. Their approach enabled multiple applications to share GPUs without modification, demonstrating throughput improvements while maintaining fairness guarantees. Building on this foundation, Sengupta *et al*. (2014) introduced the Strings scheduler, which decomposes GPU scheduling into device-level scheduling and load-balancing components. This decomposition enables better throughput while enforcing fairness constraints in multi-tenant accelerator clouds. Similarly, Menychtas *et al*. (2014) proposed disengaged scheduling strategies using kernel-mediated timeslicing with overuse control and fair queuing to guarantee access while limiting idleness. Hu *et al*. (2016) developed DASE (Dynamical Application Slowdown Estimation), a model for accurately estimating per-application GPU slowdowns under spatial multitasking. Their DASE-Fair SM allocation policy minimizes system unfairness by intelligently distributing streaming multiprocessors (SMs) among concurrent applications. This work highlighted the importance of slowdown-aware scheduling in achieving fairness objectives.

2.3 Fairness Mechanisms

Fairness in GPU resource allocation has received significant attention due to the substantial performance variations experienced by applications under different sharing scenarios. Hong *et al*. (2017a) implemented FairGV, a trap-less GPU virtualization architecture employing weighted fair queuing and work-conserving GPU-centric co-scheduling. Their system achieved near-ideal weighted fairness with strong performance isolation for mixed workloads, demonstrating that carefully designed fairness mechanisms need not sacrifice efficiency. Goswami *et al*. (2016) developed GPUShare, middleware enabling fine-grained GPU time-slicing through kernel yielding combined with central scheduling to smooth tenant share disparities. Experimental results showed improved fairness and aggregate performance compared to driver-level scheduling. The middleware approach proved particularly effective for cloud environments where application modification is impractical. Yu *et al*. (2018) introduced SMGuard, a flexible framework employing a capacity-based resource model (CapSM) with quotas, reservations, and dynamic eviction mechanisms.

SMGuard provides QoS guarantees for latency-sensitive workloads co-located with batch jobs while boosting overall utilization. This capacity-based approach represents an important advancement in balancing competing objectives of fairness, performance, and efficiency.

2.4 Container-Based GPU Sharing

The rise of container orchestration platforms has necessitated new approaches to GPU sharing tailored for containerized environments. Gu *et al*. (2018) proposed GaiaGPU, which partitions physical GPUs into virtual GPUs for containers using elastic and dynamic allocation to share GPU memory and compute resources with low overhead. Their work demonstrated the feasibility of container-native GPU virtualization without significant performance penalties. Oh *et al*. (2018) developed an adaptive fair-share method for container-based clusters enabling GPU sharing without memory shortage issues. Their approach showed improvements in both execution time and GPU memory utilization compared to static allocation methods. The adaptive nature of the scheduler allowed dynamic adjustment to varying workload characteristics, a critical capability for production environments. Hu *et al*. (2018) addressed GPU scheduling specifically for deep neural network serving through Olympian, which extends TensorFlow Serving with low-overhead profiling and interleaving to enforce fair shares across concurrent large models. Olympian achieves millisecond-scale switching with minimal overhead, demonstrating that inference serving scenarios benefit from specialized scheduling approaches distinct from training workloads.

2.5 Multi-Tenant Cloud Architectures

The architectural integration of GPU management within multi-tenant cloud platforms presents unique challenges. Gupta *et al*. (2011) introduced Pegasus, implementing hypervisor-level scheduling methods treating accelerators as first-class schedulable resources. Their work enabled fair and efficient GPU sharing across virtual machines, establishing foundational principles for cloud-based GPU resource management. Margiolas and O'Boyle (2016) developed portable software-managed scheduling on accelerators using runtime and JIT compilation to enable transparent, portable scheduling control. Their approach improved fairness and throughput across thousands of diverse workloads without requiring application-specific tuning. Tanasic *et al*. (2014) proposed hardware extensions enabling preemptive multitasking with dynamic SM distribution, improving responsiveness and fairness for multiprogrammed GPU workloads. Patchamatla (2018) examined the optimization of Kubernetes-based multi-tenant container environments in OpenStack specifically for scalable AI workflows. This work identified GPU sharing and advanced scheduling as critical but under-explored challenges, highlighting the need for dynamic resource allocation strategies and GPU-aware scheduling within Kubernetes-OpenStack deployments. The research established the foundation upon which this current investigation builds.

3. Architectural Components of Kubernetes–OpenStack GPU Management

3.1 Kubernetes Device Plugin Framework

Kubernetes provides a device plugin framework enabling vendor-specific resource management for specialized hardware including GPUs. Device plugins run as daemonsets on each node, advertising available device resources to the kubelet and facilitating device allocation during pod scheduling. The plugin interface supports device discovery, health monitoring, and allocation, but does not inherently provide fairness guarantees or advanced scheduling policies (Patchamatla, 2018). The standard Kubernetes scheduler treats GPUs as extended resources, allocating them based on simple quantity matching without awareness of workload characteristics, GPU utilization patterns, or fairness objectives. This limitation necessitates custom scheduler extensions or external scheduling components to implement intelligent GPU allocation policies (Hong *et al.,* 2017b). Device plugins must coordinate with these enhanced schedulers to translate high-level scheduling decisions into concrete device assignments.

3.2 OpenStack Nova and Ironic GPU Management

OpenStack's compute service (Nova) manages GPU resources through flavor extra specifications and PCI passthrough mechanisms. Nova scheduler filters and weighers determine placement of instances on compute nodes based on available GPU resources. The Ironic bare-metal service extends this capability to physical servers, enabling direct GPU access without virtualization overhead (Patchamatla, 2018). Integration between Kubernetes and OpenStack typically occurs through the OpenStack Cloud Provider, which enables Kubernetes to consume OpenStack resources including GPU-equipped instances. However, this integration layer does not automatically inherit GPU-aware scheduling capabilities from either platform, requiring additional coordination mechanisms to achieve intelligent GPU allocation (Gu *et al.,* 2018).

3.3 Integration Architecture

The integration of Kubernetes device plugins with OpenStack GPU management creates a multi-layered scheduling architecture. At the infrastructure layer, OpenStack Nova determines instance placement on GPU-equipped compute nodes. At the orchestration layer, Kubernetes schedules pods onto these instances based on resource requests and constraints. At the device layer, device plugins manage GPU allocation to individual containers within pods. This

hierarchical structure introduces coordination challenges. Scheduling decisions at one layer may conflict with constraints or objectives at another layer. For example, OpenStack may optimize for compute node utilization while Kubernetes optimizes for pod affinity, potentially leading to suboptimal GPU allocation (Patchamatla, 2018). Effective intelligent scheduling requires cross-layer coordination mechanisms that propagate fairness policies and workload characteristics throughout the stack. Table 1 summarizes the key components and their responsibilities in the Kubernetes–OpenStack GPU management architecture.

Table 1: Architectural Components and Responsibilities

Component	Layer	Primary Responsibilities	GPU Management Capabilities
Kubernetes Device Plugin	Device	GPU discovery, health monitoring, allocation	Device-level resource tracking and assignment
Kubernetes Scheduler	Orchestration	Pod placement, resource matching	Extended resource allocation based on requests
OpenStack Nova	Infrastructure	Instance placement, PCI passthrough	GPU-equipped instance scheduling and provisioning
OpenStack Ironic	Infrastructure	Bare-metal provisioning	Direct GPU access without virtualization
Cloud Provider Integration	Cross-layer	Resource synchronization	GPU resource visibility across platforms

3.4 Scheduling Decision Flow

GPU scheduling in Kubernetes–OpenStack environments follows a multi-stage decision flow. First, users specify GPU requirements in pod specifications using resource requests and limits. Second, the Kubernetes scheduler identifies candidate nodes with sufficient available GPU resources. Third, device plugins on selected nodes allocate specific GPU devices to containers. Fourth, OpenStack manages the underlying compute instances hosting these nodes, ensuring appropriate GPU hardware availability. This flow can be enhanced with custom schedulers implementing fairness-aware policies. Custom schedulers can intercept scheduling requests, evaluate fairness metrics across tenants, and make allocation decisions balancing immediate resource requests with long-term fairness objectives (Goswami *et al.,* 2016). Implementation typically involves webhook-based scheduler extenders or complete custom scheduler replacements.

4. Fairness Mechanisms and Scheduling Policies

4.1 Fairness Metrics and Objectives

Defining fairness in GPU resource allocation requires careful consideration of multiple competing objectives. Common fairness metrics include proportional fairness, where resources are allocated proportionally to tenant entitlements; max-min fairness, which maximizes the minimum allocation received by any tenant; and dominant resource fairness (DRF), which extends max-min fairness to multi-resource scenarios (Sengupta *et al.,* 2013). For AI workloads, fairness metrics must account for workload heterogeneity. Training jobs may require sustained GPU access over hours or days, while inference services need low-latency GPU availability for milliseconds at a time. A purely time-based fairness metric may disadvantage inference workloads, while allocation-count-based metrics may unfairly favor training jobs (Hu *et al.,* 2018). Effective fairness mechanisms must incorporate workload-aware metrics that account for these differences.

4.2 Priority-Based Queuing

Priority-based queuing assigns different priority levels to workloads based on tenant service-level agreements (SLAs), workload type, or other policy criteria. High-priority workloads receive preferential GPU access, potentially preempting lower-priority tasks. Hong *et al*. (2017a) demonstrated that weighted fair queuing combined with priority mechanisms achieves near-ideal fairness while accommodating differentiated service requirements. Implementation in Kubernetes environments typically leverages priority classes, which influence pod scheduling order and preemption decisions. However, standard Kubernetes priority preemption operates at the pod level, not the GPU device level, requiring coordination with device plugins to achieve fine-grained GPU-aware preemption (Oh *et al.,* 2018). Custom admission controllers can enforce policies ensuring high-priority pods receive GPU allocations even when resources are scarce.

4.3 Workload-Aware Preemption

Preemption mechanisms enable schedulers to reclaim GPU resources from running workloads to satisfy higher-priority or fairness-constrained allocations. Traditional GPU architectures lack native preemption support, requiring software-based approaches such as checkpoint-restart or cooperative yielding (Tanasic *et al.,* 2014). Checkpoint-restart saves workload state and terminates execution, later resuming from the checkpoint when resources become available. Cooperative yielding, implemented in systems like GPUShare, relies on applications periodically checking for

preemption signals and voluntarily releasing GPU resources (Goswami *et al.,* 2016). This approach minimizes overhead but requires application awareness or middleware intervention. For containerized environments, preemption can be implemented through container lifecycle hooks, enabling graceful shutdown and state preservation before GPU resource reclamation. Workload-aware preemption considers job characteristics when making preemption decisions. For instance, nearly-complete training jobs may be exempted from preemption to avoid wasting invested computation, while long-running jobs with frequent checkpointing may be preferentially preempted (Yu *et al.,* 2018). Such policies require schedulers to track workload progress and estimate completion times, adding complexity but improving overall system efficiency.

4.4 Time-Slicing and Spatial Partitioning

GPU sharing can be achieved through time-slicing, where multiple workloads take turns accessing the entire GPU, or spatial partitioning, where GPU resources are divided among concurrent workloads. Time-slicing provides strong isolation and simplifies resource accounting but introduces context-switching overhead and may increase latency for interactive workloads (Menychtas *et al.,* 2014). Spatial partitioning, implemented through techniques like streaming multiprocessor (SM) allocation or GPU memory partitioning, enables true concurrent execution. Hu *et al*. (2016) showed that intelligent SM allocation based on slowdown estimation achieves better fairness than naïve equal partitioning. However, spatial partitioning requires careful management to prevent memory contention and ensure performance isolation (Kato *et al.,* 2012). Hybrid approaches combining time-slicing and spatial partitioning offer flexibility to accommodate diverse workload mixes. Small inference tasks may share GPU resources spatially while large training jobs receive time-sliced exclusive access. Gu *et al*. (2018) demonstrated that dynamic switching between sharing modes based on workload characteristics improves both utilization and fairness compared to static approaches.

4.5 Capacity-Based Resource Models

Capacity-based resource models abstract GPU capabilities into measurable units enabling fine-grained allocation and accounting. Yu *et al*. (2018) introduced CapSM, modeling GPU capacity in terms of streaming multiprocessor availability and memory bandwidth. This model enables schedulers to enforce quotas and reservations with greater precision than device-count-based approaches. Capacity models facilitate QoS guarantees by reserving specific capacity levels for latency-sensitive workloads while allowing best-effort workloads to consume remaining capacity. Dynamic eviction mechanisms can reclaim capacity from best-effort tasks when reserved capacity is needed, ensuring SLA compliance (Yu *et al.,* 2018). Implementation requires runtime monitoring of GPU utilization metrics and dynamic adjustment of capacity allocations based on observed demand. Table 2 compares different fairness mechanisms across key dimensions relevant to multi-tenant AI workloads.

Table 2: Comparison of Fairness Mechanisms

Mechanism	Isolation Quality	Overhead	Workload Suitability	Implementation Complexity	Fairness Granularity
Priority Queuing	Medium	Low	Mixed workloads	Low	Coarse (tenant-level)
Time-Slicing	High	Medium	Batch training	Medium	Fine (job-level)
Spatial Partitioning	Medium	Low	Concurrent inference	High	Fine (resource-level)
Workload-Aware Preemption	Medium-High	Medium-High	Heterogeneous mixes	High	Fine (job-level)
Capacity-Based Allocation	High	Medium	QoS-sensitive workloads	High	Very Fine (capacity-level)

5. Performance Implications and Trade-offs

5.1 Impact on Training Latency

GPU scheduling policies significantly impact training latency, defined as the time required to complete model training to a target accuracy. Exclusive GPU allocation minimizes training latency by providing uninterrupted access, but achieves poor utilization when training jobs exhibit bursty GPU usage (Becchi *et al.,* 2012). Conversely, aggressive sharing through fine-grained time-slicing increases training latency due to context-switching overhead and reduced effective GPU availability. Hong *et al*. (2017a) demonstrated that fair queuing with work-conserving scheduling achieves training latencies within 5-10% of exclusive allocation while significantly improving overall GPU utilization. Work-conserving policies ensure that GPUs remain busy whenever pending work exists, avoiding idle periods that waste capacity. For training workloads tolerant of modest latency increases, such approaches offer attractive trade-offs. Preemption-based fairness mechanisms introduce additional latency through checkpoint-restart overhead. The impact depends on checkpoint frequency and state size. Frequent checkpointing reduces lost work upon preemption but

increases I/O overhead during normal execution (Tanasic *et al.,* 2014). Optimal checkpoint intervals balance these competing concerns, typically ranging from minutes to hours depending on workload characteristics.

5.2 Impact on Inference Throughput

Inference serving presents distinct performance requirements compared to training, emphasizing low latency and high throughput for individual requests. Hu *et al*. (2018) showed that specialized scheduling for DNN serving, incorporating low-overhead profiling and interleaving, maintains inference latencies below service-level objectives while supporting concurrent model execution. Their approach achieved millisecond-scale GPU switching, essential for meeting strict latency targets. Spatial partitioning proves particularly effective for inference workloads, enabling multiple models to execute concurrently on different GPU resources. However, memory contention can degrade throughput when concurrent models exceed available GPU memory, requiring careful capacity planning (Kato *et al.,* 2012). Adaptive mechanisms that dynamically adjust concurrency levels based on observed latencies help maintain QoS guarantees. Batch inference, where multiple requests are grouped for collective processing, amortizes GPU invocation overhead and improves throughput. However, batching introduces queuing delays that increase individual request latency. Schedulers must balance batch sizes against latency constraints, potentially using different batching strategies for latency-sensitive versus throughput-oriented inference workloads (Margiolas & O'Boyle, 2016).

5.3 Cost Efficiency Considerations

Cost efficiency in multi-tenant GPU infrastructures depends on utilization rates, performance delivered per dollar spent, and the value generated for tenants. Underutilized GPUs waste capital expenditure and operational costs, while oversubscription leading to performance degradation reduces tenant satisfaction and potentially violates SLAs (Patchamatla, 2018). Fair sharing mechanisms improve cost efficiency by increasing utilization without proportionally degrading performance. Goswami *et al*. (2016) reported utilization improvements of 40-60% through middleware-based GPU sharing while maintaining acceptable performance levels. These gains translate directly to cost reductions through better amortization of GPU investments across tenant workloads. However, fairness enforcement incurs overhead through scheduling computation, context switching, and monitoring. Menychtas *et al*. (2014) measured scheduling overheads below 5% for disengaged scheduling approaches, demonstrating that well-designed mechanisms achieve fairness without significant efficiency penalties. The key lies in balancing fairness granularity against overhead—extremely fine-grained fairness may consume excessive resources for monitoring and enforcement.

5.4 Trade-off Analysis

The fundamental trade-off in GPU scheduling involves balancing fairness, performance, and efficiency. Strict fairness enforcement may reduce overall system throughput by preventing efficient workload consolidation. Conversely, pure efficiency optimization may lead to resource starvation for some tenants, violating fairness objectives (Sengupta *et al.,* 2014). Workload heterogeneity complicates this trade-off. Mixed workloads combining training and inference exhibit different resource consumption patterns and performance sensitivities. Scheduling policies optimized for training may perform poorly for inference, and vice versa. Adaptive approaches that adjust scheduling strategies based on workload mix offer potential solutions but increase implementation complexity (Oh *et al.,* 2018). Table 3 quantifies trade-offs across different scheduling approaches based on representative research findings.

Table 3: Performance Trade-offs Across Scheduling Approaches

Scheduling Approach	Avg. Training Latency Impact	Inference Latency (P95)	GPU Utilization	Fairness Score (0-1)	Implementation Overhead
Exclusive Allocation	Baseline (1.0×)	5-10 ms	35-45%	0.3-0.4	Minimal
Simple Time-Slicing	1.3-1.5×	15-25 ms	65-75%	0.7-0.8	Low
Weighted Fair Queuing	1.05-1.1×	8-12 ms	70-80%	0.85-0.95	Medium
Spatial Partitioning	1.1-1.2×	6-9 ms	75-85%	0.75-0.85	Medium-High
Adaptive Hybrid	1.08-1.15×	7-11 ms	80-90%	0.9-0.95	High

Note: Metrics synthesized from Hong et al. (2017ª), Hu et al. (2018), Goswami et al. (2016), and Gu et al. (2018).

5.5 Tenant Satisfaction and SLA Compliance

Ultimately, scheduling effectiveness must be evaluated through tenant satisfaction and SLA compliance metrics. Tenants care about predictable performance, fair resource access relative to their entitlements, and cost-effectiveness. Scheduling policies that achieve high utilization but introduce unpredictable performance variations may reduce satisfaction despite technical efficiency (Gupta *et al.,* 2011). SLA compliance requires schedulers to provide performance guarantees, typically expressed as latency percentiles or minimum throughput levels. Capacity-based resource models with reservations enable strong SLA guarantees by isolating reserved capacity from best-effort workloads (Yu *et al.,* 2018). However, strict reservations may reduce overall utilization, creating tension between SLA compliance and efficiency objectives. Transparent fairness mechanisms that clearly communicate resource allocation policies and actual

allocations received help build tenant trust. Monitoring and reporting systems that provide visibility into GPU usage, queue positions, and fairness metrics enable tenants to understand their resource consumption and plan workloads accordingly (Sengupta *et al.,* 2013).

6. Implementation Considerations and Challenges

6.1 Device Plugin Extensions

Implementing intelligent GPU scheduling in Kubernetes requires extending the standard device plugin framework. Extensions must support additional capabilities including workload characterization, fairness metric tracking, and coordination with custom schedulers. Device plugins can expose extended attributes describing GPU capabilities, current utilization, and allocated capacity, enabling schedulers to make informed decisions (Patchamatla, 2018). Integration with container runtime interfaces (CRI) enables device plugins to intercept container creation and enforce allocation policies at the runtime level. This integration point allows implementation of time-slicing through runtime-managed GPU access control or spatial partitioning through cgroup-based resource limits (Oh *et al.,* 2018). However, CRI extensions require careful design to maintain compatibility with standard Kubernetes components.

6.2 Cross-Layer Coordination

Effective GPU scheduling in Kubernetes–OpenStack environments requires coordination across infrastructure, orchestration, and device layers. Information about tenant entitlements, workload characteristics, and fairness metrics must propagate throughout the stack. This coordination can be achieved through shared metadata services, message queues, or distributed consensus mechanisms (Gu *et al.,* 2018). OpenStack metadata services provide one integration point, allowing Kubernetes schedulers to query tenant quotas and entitlements defined in OpenStack. Conversely, Kubernetes can report actual resource consumption back to OpenStack for billing and capacity planning purposes. Bidirectional information flow ensures consistency between platform layers and enables unified resource management (Patchamatla, 2018).

6.3 Monitoring and Profiling

Intelligent scheduling requires comprehensive monitoring of GPU utilization, workload performance, and fairness metrics. Monitoring systems must collect fine-grained metrics including SM utilization, memory bandwidth consumption, kernel execution times, and context-switch frequencies. These metrics inform scheduling decisions and enable adaptive policy adjustments (Hu *et al.,* 2018). Profiling infrastructure characterizes workload resource consumption patterns, enabling workload-aware scheduling. Lightweight profiling techniques that impose minimal overhead are essential for production deployments. Hu *et al*. (2018) demonstrated profiling overhead below 2% through strategic sampling and offline analysis. Profiling data can be cached and reused for recurring workloads, amortizing collection costs.

6.4 Policy Configuration and Management

Fairness policies must be configurable to accommodate diverse tenant requirements and organizational objectives. Policy management systems should support declarative policy specifications, enabling administrators to define fairness objectives, priority levels, and resource reservations without low-level scheduler modifications (Yu *et al.,* 2018). Policy languages can express complex rules such as "guarantee 30% GPU capacity to tenant A during business hours with best-effort access otherwise." Policy validation and simulation tools help administrators understand policy implications before deployment. Simulation using historical workload traces reveals potential fairness violations or performance degradations, enabling policy refinement (Sengupta *et al.,* 2014). Continuous policy evaluation in production environments detects drift between intended and actual fairness outcomes, triggering alerts when corrections are needed.

6.5 Fault Tolerance and Resilience

GPU scheduling systems must handle failures gracefully, including device failures, node failures, and scheduler failures. Checkpointing mechanisms enable workload recovery after failures, though checkpoint overhead must be balanced against recovery time objectives (Tanasic *et al.,* 2014). Distributed scheduler architectures with leader election provide fault tolerance for the scheduling control plane. Device-level fault detection requires monitoring GPU health metrics and detecting degraded performance indicative of hardware issues. Automatic workload migration from unhealthy GPUs to healthy alternatives maintains service availability (Kato *et al.,* 2012). However, migration introduces temporary performance disruptions and requires careful coordination to preserve fairness properties during the transition.

6.6 Security and Isolation

Multi-tenant GPU sharing raises security concerns including information leakage through shared memory, side-channel attacks exploiting execution timing, and denial-of-service through resource exhaustion. Strong isolation mechanisms are essential to prevent malicious or buggy workloads from impacting co-located tenants (Becchi *et al.,* 2012). Memory isolation can be enforced through GPU virtual memory mechanisms that prevent unauthorized access to

other tenants' memory regions. Execution isolation limits the impact of runaway kernels through timeout mechanisms and resource quotas (Kato *et al.,* 2012). However, complete isolation may conflict with efficiency objectives, as strict partitioning reduces opportunities for statistical multiplexing and dynamic resource sharing.

7. CONCLUSION AND FUTURE DIRECTIONS

This paper has examined intelligent GPU scheduling and fairness mechanisms for multi-tenant AI workloads in Kubernetes–OpenStack environments, building upon the foundational work of Patchamatla (2018) on optimizing containerized multi-tenant infrastructures. The analysis reveals that effective GPU resource management requires careful integration of scheduling policies across infrastructure, orchestration, and device layers, with explicit consideration of fairness objectives alongside performance and efficiency goals. Key findings indicate that hybrid scheduling approaches combining time-slicing with spatial partitioning, guided by workload-aware policies, achieve superior outcomes compared to static allocation schemes. Weighted fair queuing mechanisms, as demonstrated by Hong *et al*. (2017a), provide near-ideal fairness with minimal performance overhead. Capacity-based resource models, exemplified by SMGuard (Yu *et al.,* 2018), enable fine-grained QoS guarantees essential for heterogeneous workload mixes combining latency-sensitive inference with throughput-oriented training.

The architectural integration of Kubernetes device plugins with OpenStack Nova and Ironic GPU management presents both opportunities and challenges. While the layered architecture enables flexible deployment models, it requires careful coordination to propagate fairness policies and workload characteristics throughout the stack. Cross-layer information sharing through metadata services and monitoring infrastructure proves essential for maintaining consistency and enabling informed scheduling decisions. Performance trade-offs between fairness, efficiency, and tenant satisfaction necessitate adaptive scheduling approaches that adjust policies based on workload characteristics and system state. Workload-aware preemption, dynamic capacity allocation, and priority-based queuing provide mechanisms for balancing competing objectives. However, implementation complexity and operational overhead increase with policy sophistication, requiring careful cost-benefit analysis. Future research directions include several promising areas. First, machine learning-based scheduling policies that learn optimal allocation strategies from historical workload patterns could improve upon hand-crafted heuristics. Second, integration with emerging GPU architectures supporting hardware-level multi-tenancy and preemption may enable more efficient sharing with stronger isolation. Third, extension to heterogeneous accelerator environments incorporating TPUs, FPGAs, and specialized AI chips would broaden applicability beyond GPU-centric infrastructures. Fourth, development of standardized fairness metrics and benchmarks for multi-tenant GPU systems would facilitate objective comparison of scheduling approaches and drive community progress. Fifth, investigation of economic mechanisms such as auction-based allocation or dynamic pricing could align resource allocation with tenant value generation rather than purely technical metrics. Finally, exploration of federated scheduling approaches enabling GPU sharing across multiple Kubernetes clusters and OpenStack regions would address increasingly distributed AI workload patterns.

The proliferation of AI applications ensures continued growth in GPU demand and intensifying pressure on multi-tenant infrastructures to deliver fair, efficient resource allocation. Intelligent scheduling mechanisms incorporating workload awareness, adaptive fairness policies, and cross-layer coordination represent essential capabilities for next-generation cloud platforms supporting scalable AI workflows. This research contributes to the growing body of knowledge on GPU resource management in containerized environments and provides practical insights for deploying production-grade multi-tenant AI infrastructures.

REFERENCES

- Becchi, M., Sajjapongse, K., Graves, I., Procter, A., Ravi, V., & Chakradhar, S. (2012). A virtual memory based runtime to support multi-tenancy in clusters with GPUs. *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, 97–108. https://doi.org/10.1145/2287076.2287090
- Chiobi, N. F. (2016). Integrating geospatial analytics and business intelligence for workflow optimization in pharmaceutical supply chains. *Scholars Journal of Economics, Business and Management, 3*(12), 709–723. https://doi.org/10.36347/sjebm.2016.v03i12.009
- Goswami, A., Young, J., Schwan, K., Farooqui, N., Gavrilovska, A., Wolf, M., & Eisenhauer, G. (2016). GPUShare: Fair-sharing middleware for GPU clouds. *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops*, 629–638. https://doi.org/10.1109/IPDPSW.2016.94
- Gu, J., Song, S., Li, Y., Luo, H., Qian, D., & Yuan, C. (2018). GaiaGPU: Sharing GPUs in container clouds. *Proceedings of the 2018 IEEE Intl Conference on Parallel & Distributed Processing with Applications*, 469–476. https://doi.org/10.1109/BDCLOUD.2018.00077
- Gupta, V., Gavrilovska, A., Schwan, K., Kharche, H., Tolia, N., Talwar, V., & Ranganathan, P. (2011). GViM: GPU-accelerated virtual machines. *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*, 17–24.

- Hong, C.-H., Spence, I., & Nikolopoulos, D. S. (2017a). FairGV: Fair and fast GPU virtualization. *IEEE Transactions on Parallel and Distributed Systems*, *28*(12), 3472–3485. https://doi.org/10.1109/TPDS.2017.2717908
- Hong, C.-H., Spence, I., & Nikolopoulos, D. S. (2017b). GPU virtualization and scheduling methods: A comprehensive survey. *ACM Computing Surveys*, *50*(3), Article 35. https://doi.org/10.1145/3068281
- Hu, Q., Shu, J., Fan, J., & Lu, Y. (2016). Run-time performance estimation and fairness-oriented scheduling policy for concurrent GPGPU applications. *Proceedings of the 2016 45th International Conference on Parallel Processing*, 57–66. https://doi.org/10.1109/ICPP.2016.14
- Hu, Y., Rallapalli, S., Ko, B., Govindan, R., & Srivastava, M. (2018). Olympian: Scheduling GPU usage in a deep neural network model serving system. *Proceedings of the 19th International Middleware Conference*, 53–66. https://doi.org/10.1145/3274808.3274813
- Joseph, C. (2013). From fragmented compliance to integrated governance: A conceptual framework for unifying risk, security, and regulatory controls. *Scholars Journal of Engineering and Technology, 1*(4), 238–250.
- Kato, S., McThrow, M., Maltzahn, C., & Brandt, S. A. (2012). Gdev: First-class GPU resource management in the operating system. *Proceedings of the 2012 USENIX Annual Technical Conference*, 401–412.
- Margiolas, C., & O'Boyle, M. (2016). Portable and transparent software managed scheduling on accelerators for fair resource sharing. *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, 82–93. https://doi.org/10.1145/2854038.2854040
- Menychtas, K., Shen, K., & Scott, M. L. (2014). Disengaged scheduling for fair, protected access to fast computational accelerators. *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, 301–316.
- Oh, J., Kim, S., & Kim, Y. (2018). Toward an adaptive fair GPU sharing scheme in container-based clusters. *Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing*, 476–479. https://doi.org/10.1109/FAS-W.2018.00029
- Patchamatla, P. S. (2018). Optimizing Kubernetes-based multi-tenant container environments in OpenStack for scalable AI workflows. *International Journal of Advanced Research in Education and Technology*, *5*(3). https://doi.org/10.15680/ijarety.2018.0503002
- Sengupta, D., Belapure, R., & Schwan, K. (2013). Multi-tenancy on GPGPU-based servers. *Proceedings of the 6th International Systems and Storage Conference*, Article 3. https://doi.org/10.1145/2465829.2465830
- Sengupta, D., Goswami, A., Schwan, K., & Pallavi, K. (2014). Scheduling multi-tenant cloud workloads on accelerator-based systems. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 513–524.
- Tanasic, I., Gelado, I., Cabezas, J., Ramirez, A., Navarro, N., & Valero, M. (2014). Enabling preemptive multiprogramming on GPUs. *Proceedings of the 41st Annual International Symposium on Computer Architecture*, 193–204.
- Yu, C., Bai, Y., Yang, H., Cheng, K., & Yuhao, G. (2018). SMGuard: A flexible and fine-grained resource management framework for GPUs. *IEEE Transactions on Parallel and Distributed Systems*, *29*(12), 2681–2694. https://doi.org/10.1109/TPDS.2018.2848621