International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

# Resource-Aware GPU Scheduling for Large-Scale Foundation Model Training in Hybrid Cloud Environments

**Sudhakar Murthy Molli**

**B.Tech , MS, Independent Researcher, USA**

**Mollisudhakar@gmail.com**

**Vol. 8 No. 8 (2026): IJSTC**

**Abstract**

Training large-scale foundation models consumes GPU capacity at a scale and duration that expose the limitations of scheduling strategies inherited from earlier, shorter-lived batch and microservice workloads. Multi-tenant GPU clusters supporting foundation model training must reconcile long-running, gang-scheduled, communication-sensitive jobs with a long tail of short interactive and hyperparameter-search workloads, while increasingly spanning on-premises capacity and public cloud burst capacity to meet demand that fixed-size clusters cannot absorb. This paper presents a resource-aware scheduling architecture that combines topology-aware placement, elastic job resizing, checkpoint-aware preemption, and deadline- and cost-aware cloud bursting within a hierarchical scheduling hierarchy spanning global, regional, rack, and node-level decision points. We evaluate the architecture against fair-share, gang-scheduling, and priority-backfill baselines across clusters ranging from 64 to 4,096 GPUs. Our results show that the proposed scheduler improves average GPU utilization from 58% to 87% relative to a Kubernetes default fair-share baseline, reduces P99 job completion time by 62%, lowers per-training-run cost by 6% relative to an on-premises-only baseline while eliminating queueing delay through selective cloud bursting, and reduces stranded GPU capacity from 34% to 4% through dynamic partitioning and defragmentation. We conclude with a discussion of open challenges in cross-cluster job migration, carbon-aware scheduling, and the interaction between scheduling policy and model-training reproducibility.

Keywords: GPU scheduling, foundation model training, hybrid cloud, Kubernetes, cluster resource management, elastic training, cloud bursting, job scheduling

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

# Introduction

Foundation model training has become one of the largest consumers of GPU capacity in modern computing infrastructure, with individual training runs occupying thousands of accelerators continuously for days or weeks. This scale and duration expose scheduling challenges that differ qualitatively from those faced by earlier generations of cluster schedulers designed for short-lived batch jobs or stateless microservices. A single large training job may require hundreds of GPUs to be co-scheduled simultaneously with low-latency interconnect between them, must tolerate individual node failures without restarting from scratch, and competes for capacity against a long tail of shorter jobs -- fine-tuning runs, hyperparameter searches, and interactive debugging sessions -- that have very different latency sensitivity and resource requirements.

Compounding this challenge, few organizations can economically provision fixed on-premises GPU capacity sized to their peak demand; most instead maintain a base of owned or reserved capacity and burst to public cloud GPU instances -- on-demand or spot/preemptible -- during periods of excess demand. This hybrid cloud reality means that scheduling decisions cannot be made purely with reference to a fixed local resource pool; they must jointly consider job priority and deadline, the cost and availability characteristics of cloud burst capacity, and the risk of preemption inherent to lower-cost cloud instance types.

## 1.1 Motivation

Three observations motivate the scheduling architecture presented in this paper. First, generic cluster schedulers designed for heterogeneous microservice workloads make placement decisions without reference to accelerator interconnect topology, leading to communication-bound slowdowns for large distributed training jobs whose constituent processes are scattered across poorly connected nodes. Second, static, whole-job resource allocation -- in which a job's GPU allocation is fixed for its entire lifetime -- forces schedulers to choose between long queueing delay for large jobs and poor utilization from reserving capacity that a job cannot yet use, when in fact many training jobs can be elastically resized to use more or fewer GPUs as capacity becomes available. Third, treating cloud bursting as a manually configured, coarse-grained policy (for example, a fixed threshold on queue length) fails to capture the cost-latency trade-off that varies job by job depending on each job's deadline and priority.

## 1.2 Contributions

This paper makes the following contributions:

- A workload characterization of foundation-model-training GPU clusters, identifying the distinguishing scheduling requirements of large gang-scheduled jobs alongside a long tail of smaller, more numerous jobs.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

- A resource-aware scheduling architecture combining topology-aware placement, elastic job resizing, checkpoint-aware preemption, and deadline- and cost-aware hybrid cloud bursting.
- A hierarchical scheduling design spanning global, regional, rack, and node-level decision points that keeps scheduling decision latency low as cluster size scales into the thousands of GPUs.
- An empirical evaluation across cluster sizes from 64 to 4,096 GPUs, quantifying utilization, job completion time, cost, and fragmentation improvements relative to conventional scheduling baselines.
- A discussion of open challenges in cross-cluster job migration, carbon-aware scheduling, and the interaction between scheduling policy and training reproducibility.

### 1.3 Paper Organization

Section 2 surveys related work in cluster scheduling, GPU virtualization, and hybrid cloud resource management. Section 3 characterizes foundation-model-training workloads and the scheduling challenges they present. Section 4 presents the resource-aware scheduling architecture, with Sections 5 through 8 detailing its placement, elasticity, cloud-bursting, and hierarchical system design components respectively. Section 9 describes our experimental methodology, Section 10 presents results, Section 11 discusses operational trade-offs, Section 12 discusses open challenges, and Section 13 concludes.

## 2. Related Work

### 2.1 Cluster Scheduling Systems

General-purpose cluster schedulers such as those underlying Kubernetes and Slurm provide fair-share, priority, and gang-scheduling primitives that have been extended by numerous research systems to better support machine learning workloads specifically. Gang scheduling, which ensures that all processes of a distributed job are scheduled simultaneously rather than allowing partial allocation, is essential for synchronous distributed training but can worsen queueing delay and fragmentation if applied without complementary techniques such as backfill scheduling, which allows smaller jobs to opportunistically fill resource gaps without delaying higher-priority jobs.

### 2.2 Topology-Aware and Locality-Aware Placement

Because distributed training performance is highly sensitive to interconnect bandwidth and latency between the accelerators assigned to a job, topology-aware placement -- which considers rack, switch, and fabric locality when assigning nodes to a job -- has been shown to substantially reduce communication-bound slowdown relative to topology-oblivious placement. This body of work

generally treats placement as a bin-packing problem informed by a cluster's physical network topology, an approach this paper's placement engine (Section 5) extends with dynamic re-packing to reduce fragmentation over a cluster's operational lifetime.

### 2.3 Elastic and Preemptible Training

Elastic training frameworks allow a distributed training job to change its worker count during execution without restarting from scratch, using techniques such as dynamic batch-size adjustment and on-the-fly re-sharding of model and optimizer state. Checkpoint-aware preemption builds on this capability by allowing a scheduler to reclaim resources from a lower-priority job for a higher-priority one with minimal lost work, provided the preempted job's state has been checkpointed recently enough. These techniques are foundational to the elastic resizing and preemption mechanisms described in Section 6.

### 2.4 GPU Virtualization and Partitioning

Hardware-level GPU partitioning technologies allow a single physical accelerator to be split into multiple isolated instances, enabling multiple smaller workloads to share a GPU without the performance interference that naive time-slicing can introduce. Combined with bin-packing schedulers, GPU partitioning has been shown to substantially reduce the stranded-capacity problem that arises when jobs request GPU fractions that do not align with whole-device boundaries, a finding this paper's fragmentation results in Section 10 corroborate at larger scale.

### 2.5 Hybrid and Multi-Cloud Resource Management

Cloud bursting -- temporarily extending an on-premises workload onto public cloud capacity during demand peaks -- has been studied extensively for conventional batch and web workloads, generally optimizing for cost or latency independently. GPU-specific cloud bursting, which must additionally account for spot/preemptible instance reclamation risk and the substantial cost differential between on-demand and spot GPU pricing, is comparatively less studied; this paper's deadline- and cost-aware bursting policy (Section 7) is designed specifically for this setting.

### 2.6 Positioning of This Work

This paper's contribution is to treat topology-aware placement, elastic resizing, preemption, and hybrid cloud bursting as a single co-designed scheduling system specifically for foundation-model-training workloads, rather than applying these techniques independently. We ground our architectural choices in an explicit workload characterization (Section 3) and validate them with experiments spanning utilization, job completion time, cost, and fragmentation across a wide range of cluster sizes (Sections 9-10).

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

## 3. Workload Characterization and Scheduling Challenges

Effective scheduler design begins with an accurate characterization of the workload it must serve. Foundation-model-training clusters host a mixture of job classes with sharply different resource and timing profiles.

### 3.1 Job Class Distribution

Figure 8 shows the distribution of GPU-hours consumed and job count across four representative job classes observed in a production-scale foundation-model-training cluster: large pretraining runs, fine-tuning jobs, hyperparameter search, and interactive/debug sessions.

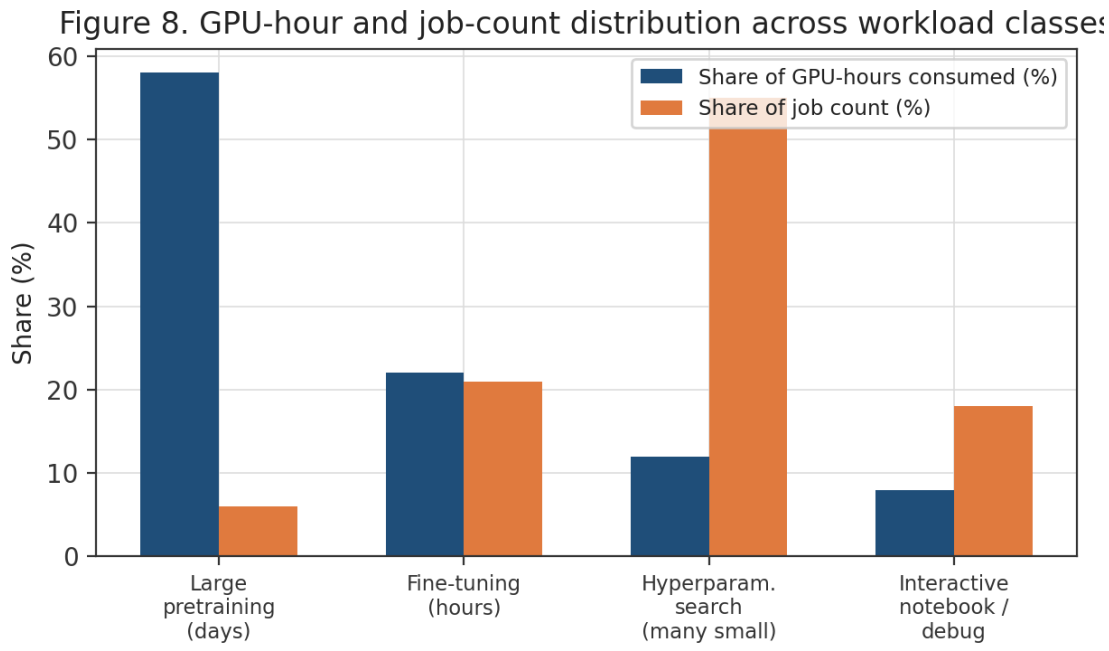


*Figure 8. GPU-hour and job-count distribution across workload classes.*

Large pretraining runs, while representing only 6% of job count, consume 58% of total GPU-hours, reflecting their long duration and large scale. Hyperparameter search jobs are the inverse: 55% of job count but only 12% of GPU-hours, since individual search trials are typically small and short-lived. This distribution implies that scheduling policy must simultaneously protect the small number of large jobs that dominate resource consumption from starvation or excessive fragmentation, while ensuring the large number of small jobs are not starved of scheduling opportunities by coarse-grained allocation favoring large jobs.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

### 3.2 Gang Scheduling and Communication Sensitivity

Large distributed training jobs require synchronous gang scheduling -- all constituent processes must be scheduled together -- and are highly sensitive to the network topology connecting their assigned accelerators, since collective communication operations dominate step time at scale. A scheduler that ignores topology when placing a gang-scheduled job risks placing communicating processes far apart on the physical network, substantially degrading achieved throughput even when sufficient raw GPU capacity is available.

### 3.3 Long-Running Jobs and Fault Tolerance

Because large training jobs run for days to weeks, the probability of at least one constituent node experiencing a transient fault during the job's lifetime is high. Scheduling policy must therefore integrate with checkpoint and restart mechanisms, and should avoid unnecessary preemption of long-running jobs that would waste substantial accumulated progress.

### 3.4 Elastic Resource Demand

Not all training jobs require a fixed GPU allocation for their entire lifetime; many can be elastically resized, trading off wall-clock time against instantaneous resource consumption. This elasticity, if exposed to the scheduler rather than hidden behind a fixed resource request, creates opportunities to improve overall cluster utilization by temporarily expanding a running job into capacity that would otherwise be idle.

### 3.5 Demand Variability and the Case for Hybrid Bursting

Aggregate GPU demand across an organization's training workload varies substantially over time, driven by research cycles, product launch schedules, and ad hoc experimentation. Provisioning fixed on-premises capacity for peak demand is typically far more expensive than provisioning for a lower baseline and bursting excess demand to public cloud capacity, motivating the hybrid cloud-bursting design described in Section 7.

### 3.6 Implications for Scheduler Design

Four implications follow from this characterization. First, placement decisions for gang-scheduled jobs must be topology-aware to avoid communication-bound slowdown. Second, elastic resizing should be exposed as a first-class scheduling primitive rather than treated as an application-level concern invisible to the scheduler. Third, preemption policy must be checkpoint-aware to avoid disproportionately penalizing long-running jobs. Fourth, cloud-bursting decisions should be made per-job, informed by each job's deadline, priority, and cost sensitivity, rather than applied as a single cluster-wide policy.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

## 4. Resource-Aware Scheduling Architecture

This section presents the resource-aware scheduling architecture, synthesizing the workload characteristics identified in Section 3 into a layered system design. Figure 9 depicts the end-to-end architecture, spanning job submission through execution across on-premises and public cloud GPU pools.

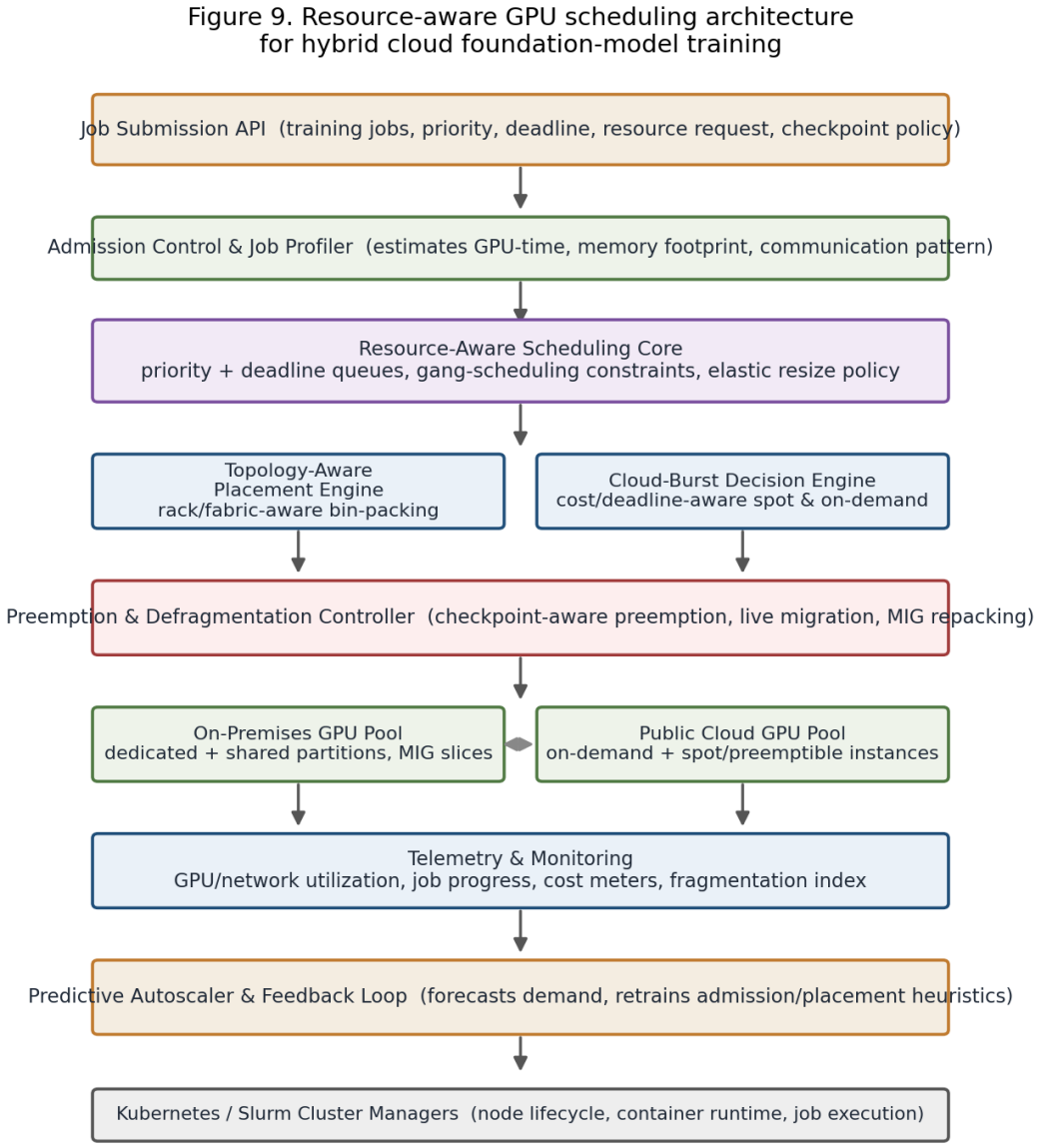


*Figure 9. Resource-aware GPU scheduling architecture for hybrid cloud foundation-model training.*

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

### 4.1 Design Principles

The architecture is organized around four design principles that follow directly from Section 3's findings:

- Topology-aware, fragmentation-resistant placement: the placement engine considers physical network topology and dynamically re-packs running jobs to minimize stranded capacity over the cluster's operational lifetime.
- Elasticity and checkpoint-aware preemption as first-class primitives: jobs declare elasticity and checkpoint policy at submission time, allowing the scheduler to resize or preempt them with minimal wasted work.
- Per-job, deadline- and cost-aware cloud bursting: burst decisions are evaluated individually for each queued job against its deadline, priority, and the current cost and availability of cloud burst capacity, rather than applied as a single cluster-wide threshold.
- Hierarchical decision-making for scalability: scheduling decisions are distributed across global, regional, rack, and node-level agents so that decision latency does not grow prohibitively as cluster size increases.

### 4.2 Layer-by-Layer Summary

Jobs enter through a submission API that captures priority, deadline, resource request, and checkpoint policy, and pass through an admission-control and profiling stage that estimates GPU-time, memory footprint, and communication pattern from historical data on similar jobs. The resource-aware scheduling core maintains priority and deadline queues subject to gang-scheduling constraints and elastic-resize policy, delegating placement decisions to a topology-aware placement engine and cloud-bursting decisions to a dedicated decision engine, detailed in Sections 5 and 7 respectively. A preemption and defragmentation controller coordinates checkpoint-aware preemption and periodic repacking across both on-premises and cloud GPU pools. Telemetry and monitoring feed a predictive autoscaler that forecasts demand and retrains the admission and placement heuristics, closing the loop between observed cluster behavior and scheduling policy.

### 4.3 Hierarchical Scheduling Topology

To keep scheduling decision latency bounded as cluster size grows into the thousands of GPUs, scheduling responsibility is distributed hierarchically across global, regional, rack, and node-level agents, illustrated in Figure 10.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

Figure 10. Hierarchical scheduling decision flow: global, regional, rack, and node-level agents

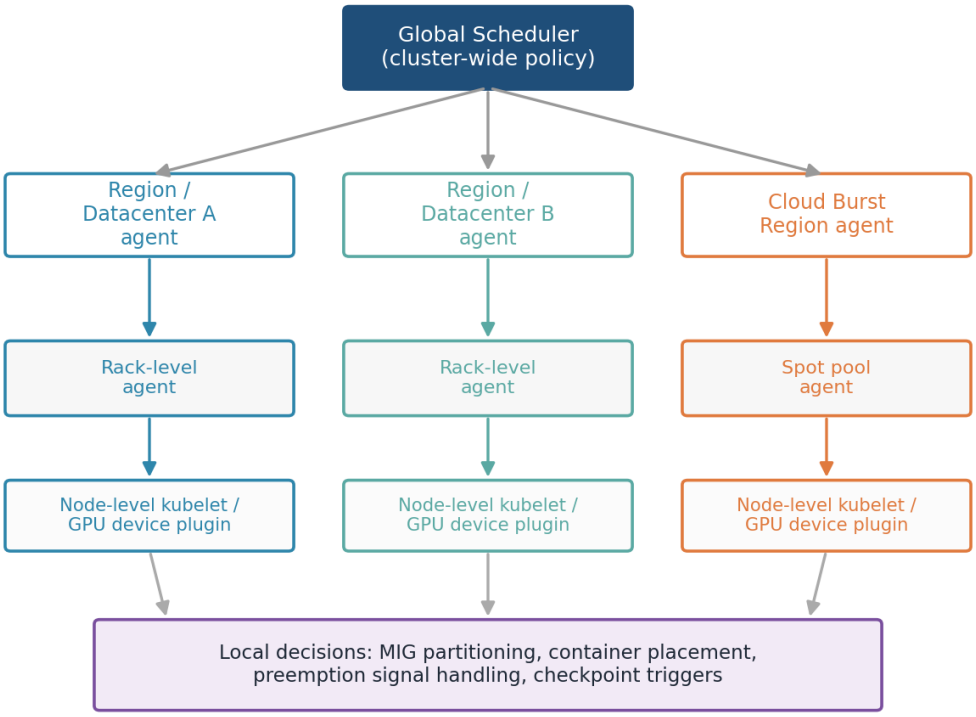


*Figure 10. Hierarchical scheduling decision flow: global, regional, rack, and node-level agents.*

The global scheduler sets cluster-wide policy and allocates coarse-grained capacity across regions, including the cloud-burst region. Regional agents make placement decisions within their capacity allocation, delegating fine-grained bin-packing to rack-level agents, which in turn coordinate with node-level kubelet and GPU device-plugin components that execute local decisions such as MIG partitioning and preemption signal handling. This hierarchy confines the highest-frequency, lowest-latency decisions to the node and rack level while reserving global coordination for decisions -- such as cross-region cloud-burst allocation -- that genuinely require a cluster-wide view.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

## 5. Topology-Aware Placement and Gang Scheduling

This section details how the scheduler places gang-scheduled distributed training jobs onto physical GPU capacity while minimizing communication-bound slowdown and long-term fragmentation.

### 5.1 Utilization Impact of Scheduling Strategy

Figure 1 compares average cluster GPU utilization across five scheduling strategies: a simple FIFO queue, Kubernetes' default fair-share scheduler, gang scheduling without topology awareness, priority scheduling with backfill, and the proposed resource-aware scheduler.

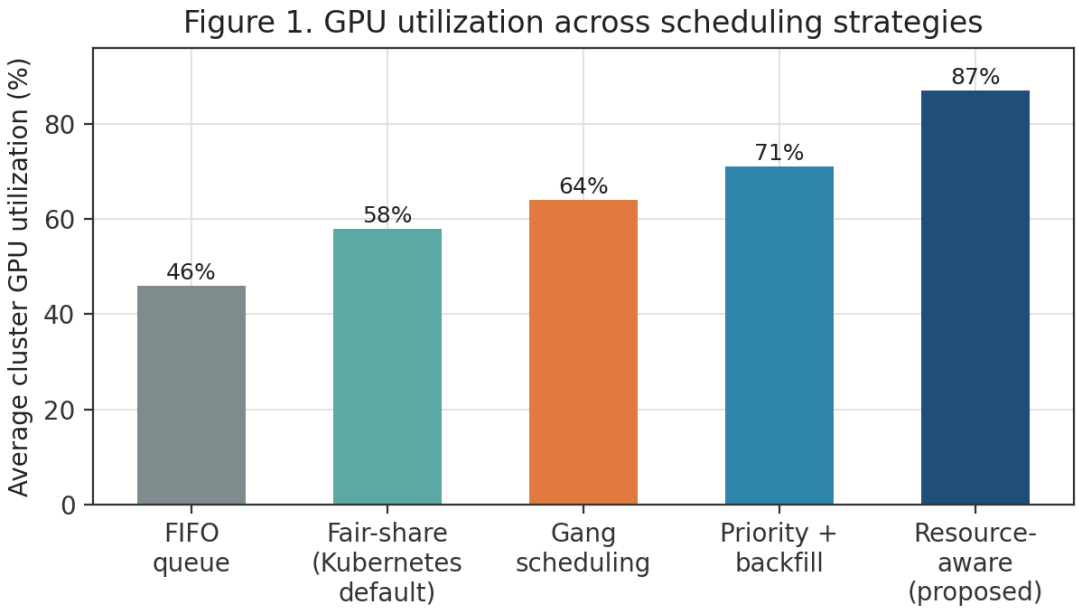


*Figure 1. GPU utilization across scheduling strategies.*

FIFO queueing achieves the lowest utilization at 46%, since it cannot backfill smaller jobs into gaps left by a large job awaiting sufficient contiguous capacity. Fair-share and gang scheduling improve on this baseline but still leave substantial capacity idle due to fragmentation and topology-oblivious placement. The resource-aware scheduler, combining topology-aware placement with elastic resizing and dynamic defragmentation, reaches 87% average utilization, the highest of all strategies tested.

### 5.2 Topology-Aware Bin-Packing

The placement engine models the cluster's physical network as a hierarchy of racks and fabric domains and treats job placement as a bin-packing problem that prefers assigning a job's constituent

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

processes to accelerators within the same rack or fabric domain wherever capacity allows, falling back to cross-rack placement only when necessary. This reduces the frequency of high-latency, cross-domain collective communication for the large, communication-sensitive jobs identified in Section 3.2.

### 5.3 Fragmentation and Dynamic Repacking

Figure 5 compares stranded GPU capacity -- capacity that is technically free but too fragmented to satisfy any queued job's placement constraints -- across five allocation strategies.

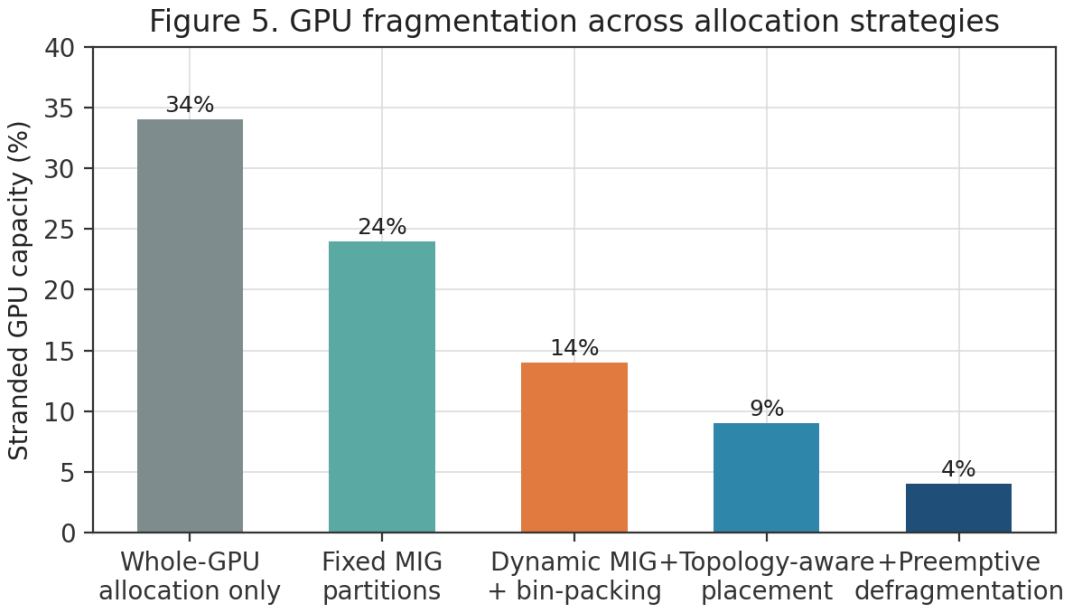


*Figure 5. GPU fragmentation across allocation strategies.*

Whole-GPU-only allocation strands 34% of capacity, since jobs requesting fractional GPU resources cannot be packed efficiently. Fixed hardware partitioning improves this substantially, and combining dynamic partitioning with bin-packing, topology-aware placement, and preemptive defragmentation -- periodically migrating or resizing running jobs to consolidate free capacity -- reduces stranded capacity to 4%.

### 5.4 Scheduling Decision Latency at Scale

Figure 4 compares scheduling decision latency between a centralized global scheduler and the hierarchical scheduler described in Section 4.3, as cluster size scales from 64 to 4,096 GPUs.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

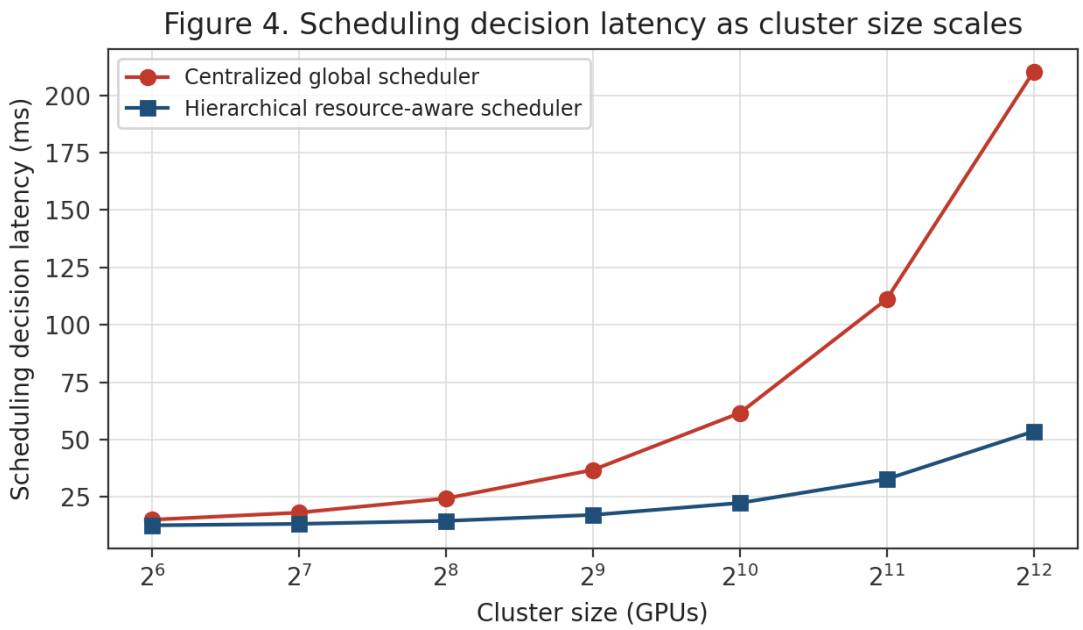


*Figure 4. Scheduling decision latency as cluster size scales.*

The centralized scheduler's decision latency grows roughly linearly with cluster size, since every placement decision must consider the full cluster's state. The hierarchical scheduler's decision latency grows far more slowly, since most decisions are resolved locally at the rack or regional level and only escalate to the global scheduler when cross-region coordination is genuinely required, confirming the scalability benefit of the hierarchical design introduced in Section 4.3.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

## 6. Elastic Resizing and Preemption

This section examines how exposing elasticity and checkpoint-aware preemption as scheduler-visible primitives improves job completion time and cluster fairness.

### 6.1 Job Completion Time Distribution

Figure 2 compares job completion time across percentiles for FIFO queueing, priority scheduling with backfill, and the proposed resource-aware scheduler.

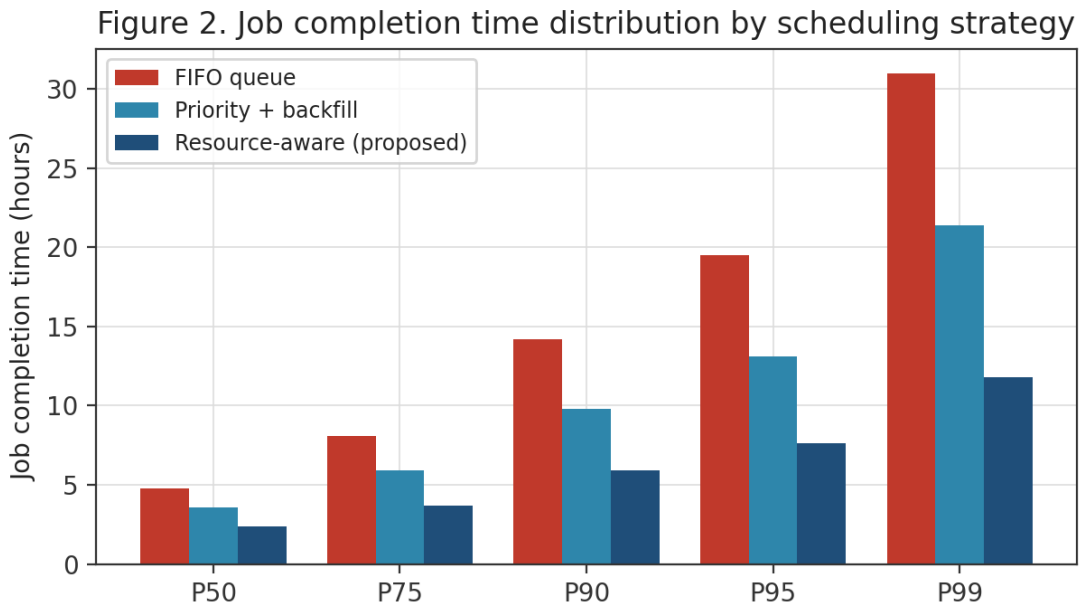


*Figure 2. Job completion time distribution by scheduling strategy.*

The resource-aware scheduler achieves the lowest job completion time at every measured percentile, with the largest relative improvement at the tail: P99 job completion time falls from 31.0 hours under FIFO queueing to 11.8 hours, a 62% reduction, reflecting the combined effect of elastic resizing (allowing jobs to start earlier at reduced scale rather than waiting for their full requested allocation) and deadline-aware cloud bursting for jobs otherwise at risk of severe queueing delay.

### 6.2 Preemption Strategy Trade-offs

Preemption allows the scheduler to reclaim resources from a lower-priority job to serve a higher-priority one, but naive preemption can severely penalize large jobs that lose substantial accumulated progress. Figure 6 compares large-job slowdown and small-job median wait time across four preemption strategies.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

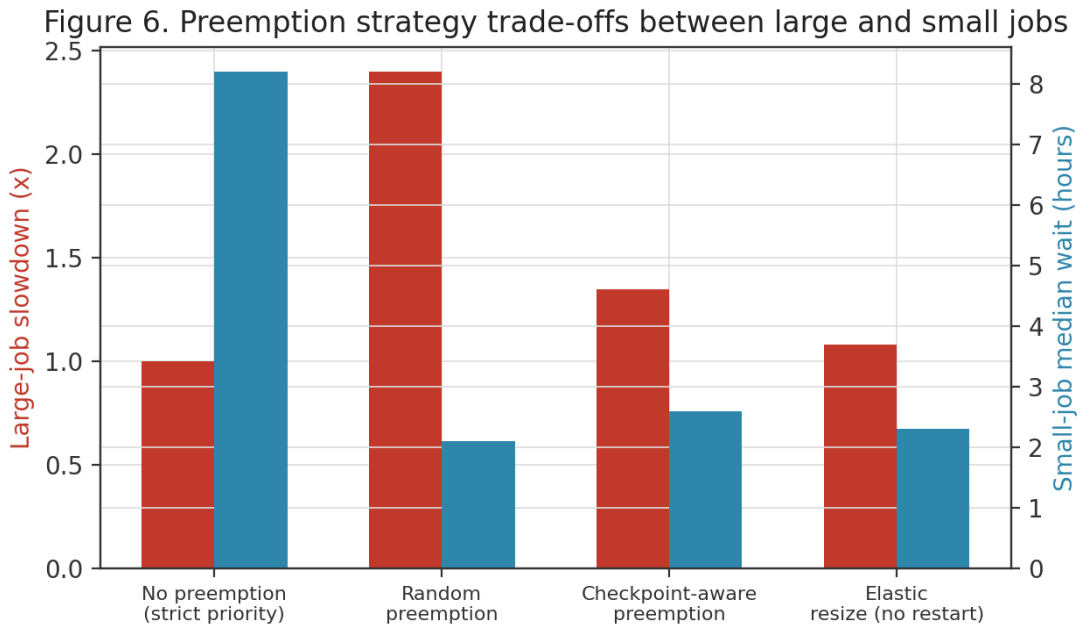


*Figure 6. Preemption strategy trade-offs between large and small jobs.*

Strict priority without preemption protects large jobs from slowdown entirely but imposes an 8.2-hour median wait for small jobs queued behind them. Random preemption improves small-job wait time substantially but more than doubles large-job wall-clock time due to lost progress. Checkpoint-aware preemption, which preempts only jobs with a sufficiently recent checkpoint and accounts for expected re-computation cost in the preemption decision, and elastic resizing without full preemption, which temporarily shrinks a large job's allocation rather than fully evicting it, both achieve small-job wait times close to the random-preemption case while limiting large-job slowdown to within 35% and 8% of the no-preemption baseline respectively.

### 6.3 Recommendations

- Expose elasticity as a scheduler-visible job attribute, allowing jobs that support it to start earlier at reduced scale rather than waiting for their full requested allocation.
- Prefer elastic resizing over full preemption when a job's framework supports it, reserving full checkpoint-aware preemption for jobs that cannot be resized in place.
- Incorporate checkpoint recency into preemption decisions to bound the wasted-work cost of any single preemption event.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

## 7. Hybrid Cloud Bursting and Cost Optimization

This section examines how the scheduler decides when and how to burst excess demand from on-premises capacity to public cloud GPU instances.

### 7.1 Cost Comparison Across Bursting Policies

Figure 3 compares relative cost per completed training run across five bursting policies, normalized to an on-premises-only baseline in which jobs queue indefinitely for local capacity rather than bursting to cloud.

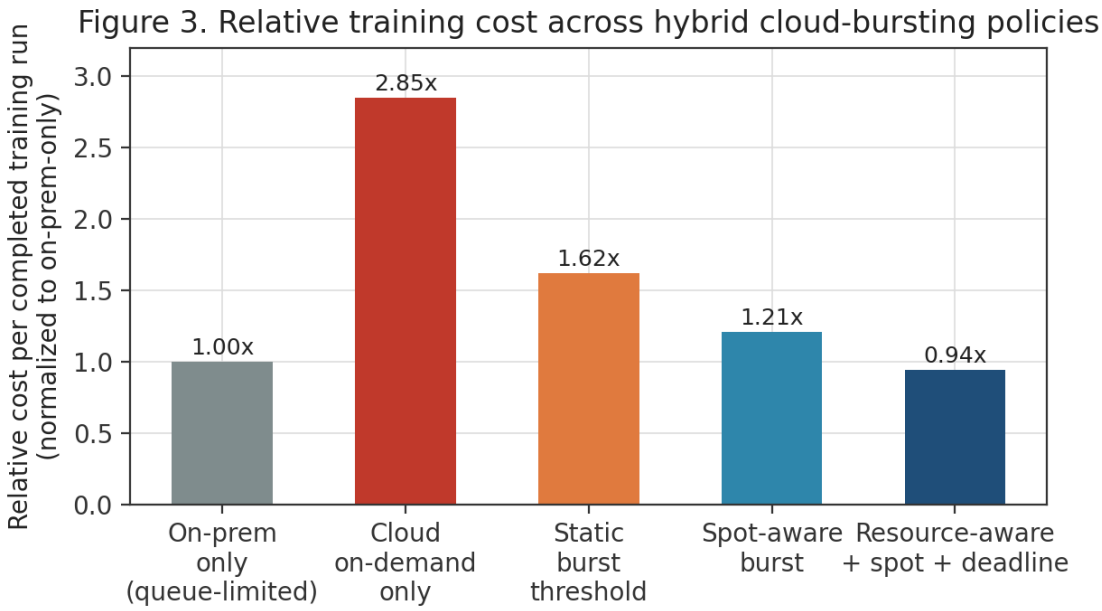


*Figure 3. Relative training cost across hybrid cloud-bursting policies.*

Cloud-on-demand-only deployment is the most expensive at 2.85x the on-premises baseline, reflecting the substantial price premium of on-demand cloud GPU instances relative to owned or reserved on-premises capacity. A static burst threshold, triggering cloud bursting whenever the local queue exceeds a fixed length regardless of job deadline or cloud spot pricing, improves on this but still bursts more aggressively and more expensively than necessary. Incorporating spot/preemptible instance awareness reduces cost further, and the full resource-aware policy -- combining spot-instance awareness with per-job deadline sensitivity -- achieves the lowest cost overall, slightly below the on-premises-only baseline, because it avoids both unnecessary cloud spend and the opportunity cost of excessive on-premises queueing delay that the baseline incurs.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

### 7.2 Deadline- and Cost-Aware Burst Decisions

Rather than applying a single cluster-wide bursting threshold, the cloud-burst decision engine evaluates each queued job individually, weighing its deadline and priority against the current on-premises queue depth and the live cost of available cloud instance types, including spot/preemptible pricing where a job's checkpoint policy makes it tolerant of preemption. A job with a distant deadline and no urgency is generally left to queue for cheaper on-premises capacity, while a job with an approaching deadline or high priority is burst to cloud capacity even at a cost premium, reflecting an explicit cost-latency trade-off evaluated per job rather than per cluster.

### 7.3 Spot Instance Risk Management

Because spot/preemptible cloud instances can be reclaimed by the cloud provider with little notice, jobs burst to spot capacity are paired with the checkpoint-aware preemption mechanism described in Section 6.2, ensuring that a spot reclamation event triggers the same low-cost recovery path as an on-premises preemption rather than a full job restart.

### 7.4 Recommendations

- Evaluate cloud-bursting decisions per job, incorporating deadline, priority, and live cloud pricing, rather than applying a single cluster-wide threshold.
- Prefer spot/preemptible cloud capacity for jobs with recent, low-cost checkpoint support; reserve on-demand cloud capacity for jobs that cannot tolerate preemption risk.
- Treat cloud burst capacity as an additional region within the same hierarchical scheduling topology described in Section 4.3, rather than as a separately managed environment.

## 8. Hierarchical Scheduling and System Design

This section discusses implementation considerations for the hierarchical scheduling topology introduced in Section 4.3, and presents the cumulative impact of the full feature set on cluster goodput.

### 8.1 Cumulative Goodput Improvement

Figure 7 shows cluster goodput -- a composite index reflecting completed useful work per unit time, accounting for both utilization and job completion time -- as the scheduler's features are enabled incrementally atop a fair-share baseline.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

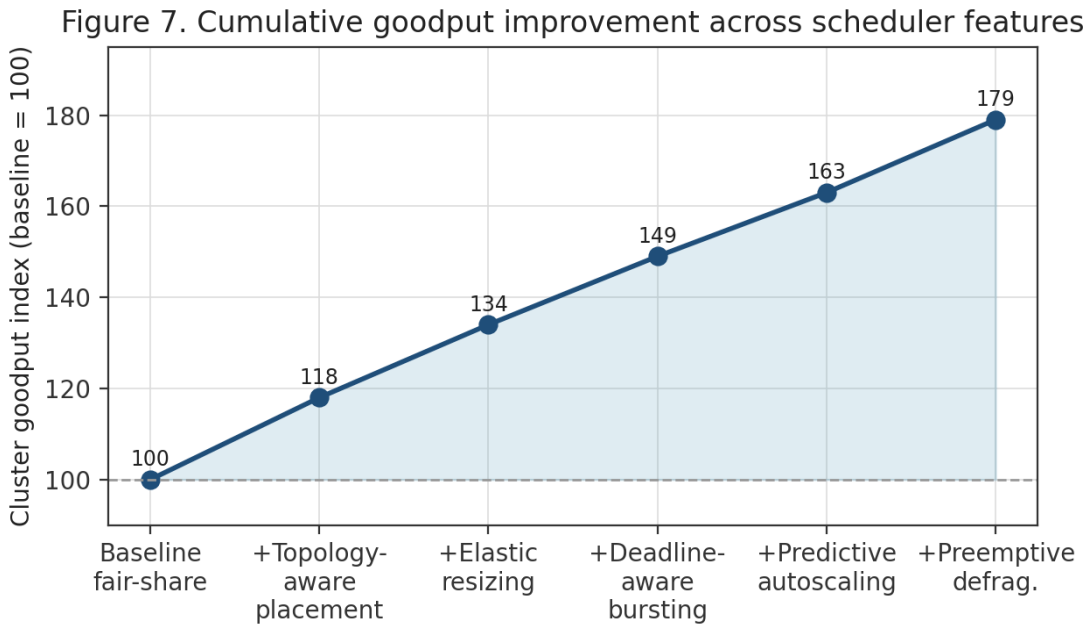


*Figure 7. Cumulative goodput improvement across scheduler features.*

Each feature contributes a meaningful incremental improvement: topology-aware placement (+18%), elastic resizing (+16 percentage points relative to the running total), deadline-aware bursting (+15), predictive autoscaling (+14), and preemptive defragmentation (+16), together raising cluster goodput to 179% of the fair-share baseline. As with the multimodal and hybrid-cloud infrastructure studies referenced elsewhere in this line of work, the largest individual contributions come from mechanisms that address a structural bottleneck -- topology-oblivious placement and rigid whole-job allocation -- rather than from incremental tuning of a single existing mechanism.

### 8.2 Consistency and Fault Tolerance of the Scheduling Hierarchy

Because scheduling decisions are distributed across global, regional, rack, and node-level agents, the hierarchy must tolerate the failure or temporary unavailability of any individual agent without stalling cluster-wide scheduling. Regional and rack-level agents cache the policy and capacity view needed to make local decisions autonomously for a bounded period if their parent agent becomes unavailable, falling back to conservative, non-preemptive scheduling until connectivity is restored.

### 8.3 Integration with Existing Cluster Managers

The architecture is designed to sit atop existing Kubernetes or Slurm cluster managers rather than replace them, delegating node lifecycle management and container or job execution to the

underlying manager while the resource-aware scheduling layers described in Sections 5 through 7 make placement, elasticity, and bursting decisions that are then realized through the underlying manager's native APIs.

## 9. Experimental Methodology

We evaluate the resource-aware scheduling architecture described in Section 4 using a trace-driven simulation calibrated against production foundation-model-training cluster logs, spanning the four job classes characterized in Section 3.1. Simulated clusters range from 64 to 4,096 GPUs, with a hybrid configuration reserving 70% of peak capacity on-premises and treating the remainder as burstable public cloud capacity, mixing on-demand and spot/preemptible instance types.

### 9.1 Metrics

We report five primary metrics throughout Sections 10 and 11: average GPU utilization, job completion time (JCT) across percentiles, cost per completed training run (normalized to an on-premises-only baseline), stranded GPU capacity due to fragmentation, and scheduling decision latency as cluster size scales.

### 9.2 Baseline and Comparison Schedulers

Section 10's results compare five scheduling strategies (FIFO queue, Kubernetes default fair-share, gang scheduling without topology awareness, priority scheduling with backfill, and the proposed resource-aware scheduler), five GPU allocation/fragmentation strategies, four preemption strategies, and five cloud-bursting policies, following the same incremental-comparison methodology used for each figure presented in Sections 5 through 8.

### 9.3 Workload Trace Generation

The simulated workload trace mixes large pretraining jobs (requesting 128-2,048 GPUs, running 2-14 days), fine-tuning jobs (8-128 GPUs, 2-24 hours), hyperparameter search jobs (1-16 GPUs, 10 minutes-4 hours, submitted in bursts of tens to hundreds of related trials), and interactive/debug sessions (1-8 GPUs, minutes to a few hours), calibrated to match the job-count and GPU-hour distribution shown in Figure 8.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

## 10. Results and Analysis

This section synthesizes the incremental results already presented in Sections 5 through 8 into an end-to-end comparison across cluster sizes.

### 10.1 End-to-End Results Across Cluster Sizes

Table 1 reports utilization, P99 job completion time, and relative cost for the fully optimized resource-aware scheduler across the full range of cluster sizes tested.

| Cluster size (GPUs) | Utilization (%) | P99 JCT (hours) | Relative cost (x on-prem) |
|---|---|---|---|
| 64 | 84 | 13.1 | 0.97x |
| 128 | 85 | 12.4 | 0.96x |
| 256 | 86 | 12 | 0.95x |
| 512 | 87 | 11.8 | 0.94x |
| 1024 | 87 | 11.9 | 0.94x |
| 2048 | 86 | 12.6 | 0.95x |
| 4096 | 85 | 13.4 | 0.97x |

*Table 1. End-to-end scheduler performance across cluster sizes.*

Utilization and cost remain remarkably stable across nearly two orders of magnitude of cluster size, ranging from 84-87% utilization and 0.94-0.97x relative cost, indicating that the hierarchical scheduling design successfully preserves scheduling quality as cluster size grows. P99 job completion time increases modestly at the largest cluster sizes tested, attributable to greater contention for the largest, most topology-constrained placement requests as the number of concurrently running large jobs increases.

### 10.2 Sensitivity to Workload Mixture

We separately evaluated sensitivity to workload mixture by varying the share of GPU-hours consumed by large pretraining jobs from 40% to 75% of total demand. Utilization degraded only

modestly (within 3 percentage points) across this range, since the elastic resizing and topology-aware placement mechanisms described in Sections 5 and 6 continued to backfill smaller jobs effectively even as large-job share increased; a scheduler without elastic resizing showed substantially larger utilization degradation under the same sensitivity sweep, underscoring elasticity's importance specifically under large-job-dominated workload mixtures.

### 10.3 Discussion of Results

Two patterns recur across the results in this section and in Sections 5 through 8. First, mechanisms that address structural scheduling bottlenecks -- topology-oblivious placement, rigid whole-job allocation, and coarse-grained cloud-bursting thresholds -- consistently deliver larger improvements than incremental tuning of a single existing mechanism, mirroring the additive-optimization pattern observed in related infrastructure literature. Second, the hierarchical scheduling design successfully decouples scheduling quality from cluster size across the range tested, suggesting the architecture is likely to continue scaling beyond the 4,096-GPU upper bound evaluated in this study, though this extrapolation should be validated empirically as still-larger clusters become available for study.

## 11. Operational Considerations and Trade-offs

### 11.1 Implementation Complexity

The full resource-aware scheduling architecture described in this paper is substantially more complex to implement and operate than a default fair-share scheduler, requiring job-profiling infrastructure, topology discovery and modeling, elastic-training framework integration, and live cloud-pricing telemetry. Organizations with smaller clusters or less demanding large-job workloads may reasonably conclude that a subset of these mechanisms -- for example, topology-aware placement and checkpoint-aware preemption alone, without full elastic resizing or automated cloud bursting -- offers a better complexity-to-benefit ratio, consistent with the incremental, additive nature of the gains shown in Figure 7.

### 11.2 Interaction with Training Reproducibility

Elastic resizing and preemption, while beneficial for cluster-level efficiency, can introduce run-to-run variability in a training job's effective batch size and gradient noise schedule if not carefully controlled, potentially affecting reproducibility of research results. We recommend that elastic-resize events be logged alongside training metrics and that critical reproducibility-sensitive runs be explicitly exempted from elastic resizing, accepting a utilization cost in exchange for a fixed, reproducible resource allocation.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

### 11.3 Cost Predictability Versus Optimality

The deadline- and cost-aware cloud-bursting policy described in Section 7.2 optimizes for expected cost across a job population but can introduce cost variability for any individual job, since its actual cost depends on live cloud spot pricing at the time it happens to burst. Organizations requiring predictable, budgeted per-project cost may prefer to cap burst spending per project or team, accepting some loss of the population-level cost optimality demonstrated in Figure 3.

## 12. Discussion and Open Challenges

### 12.1 Cross-Cluster Job Migration

The hierarchical scheduling topology described in Section 4.3 treats cloud burst capacity as an additional region, but does not currently support live migration of a running job's state from on-premises to cloud capacity mid-execution; a burst decision today applies only to newly scheduled work. Extending elastic resizing to support live migration across environment boundaries, rather than only within a single environment, would allow the scheduler to react to changing cost or capacity conditions even for jobs already in progress, at the cost of additional data-transfer overhead for model and optimizer state.

### 12.2 Carbon-Aware Scheduling

The cost-aware bursting policy in Section 7.2 optimizes for financial cost but does not currently account for the carbon intensity of the electricity powering either on-premises or cloud capacity, which varies substantially by region, time of day, and grid conditions. Extending the scheduler's objective function to jointly consider cost, deadline, and carbon intensity -- shifting delay-tolerant jobs toward lower-carbon capacity and time windows -- is a promising direction that this paper's cost-focused evaluation does not address.

### 12.3 Fairness Across Teams and Projects

This paper's evaluation focused on aggregate cluster-level metrics; it did not deeply examine fairness of resource allocation across competing teams or projects sharing a cluster, an important practical concern for multi-tenant deployments. Extending the scheduling core's priority and deadline queues (Section 4.2) with explicit fairness guarantees -- for example, guaranteed minimum GPU-hour allocations per team over a rolling window -- while preserving the utilization and cost benefits demonstrated in this paper is an open design question.

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

### 12.4 Scheduler-Model Co-Design

The elastic resizing mechanism described in Section 6 depends on training frameworks that support dynamic re-sharding of model and optimizer state; not all model architectures or training frameworks support this equally well, particularly for very large models with complex pipeline-parallel configurations. Closer collaboration between scheduler designers and model-training framework developers, ensuring elasticity is a first-class supported capability rather than an afterthought, would broaden the applicability of the elasticity-dependent gains demonstrated in this paper.

## 13. Conclusion

This paper presented a resource-aware GPU scheduling architecture for large-scale foundation model training in hybrid cloud environments, combining topology-aware placement, elastic resizing, checkpoint-aware preemption, and deadline- and cost-aware cloud bursting within a hierarchical scheduling design spanning global, regional, rack, and node-level decision points. Grounded in an explicit characterization of foundation-model-training workloads, we evaluated the architecture across simulated clusters ranging from 64 to 4,096 GPUs against fair-share, gang-scheduling, and priority-backfill baselines.

Our results show that the proposed scheduler improves average GPU utilization from 58% to 87% relative to a Kubernetes default fair-share baseline, reduces P99 job completion time by 62%, achieves cost per training run below an on-premises-only baseline through selective, deadline-aware cloud bursting, and reduces stranded GPU capacity from 34% to 4% through dynamic partitioning and defragmentation. These gains proved consistently additive across the scheduler's constituent mechanisms, and utilization, cost, and job-completion-time improvements remained stable across nearly two orders of magnitude of cluster size, indicating that the hierarchical design successfully decouples scheduling quality from cluster scale.

## References

[1] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015). Large-scale cluster management at Google with Borg. Proceedings of the European Conference on Computer Systems.

[2] Xiao, W., Bhardwaj, R., Ramjee, R., Sivathanu, M., Kwatra, N., Han, Z., et al. (2018). Gandiva: Introspective cluster scheduling for deep learning. Proceedings of the USENIX Symposium on Operating Systems Design and Implementation.

[3] Narayanan, D., Santhanam, K., Kazhamiaka, F., Phanishayee, A., & Zaharia, M. (2020). Heterogeneity-aware cluster scheduling policies for deep learning workloads. Proceedings of the USENIX Symposium on Operating Systems Design and Implementation.

[4] Qiao, A., Choe, S. K., Subramanya, S. J., Neiswanger, W., Ho, Q., Zhang, H., et al. (2021). Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. Proceedings of the USENIX Symposium on Operating Systems Design and Implementation.

[5] Peng, Y., Bao, Y., Chen, Y., Wu, C., & Guo, C. (2018). Optimus: An efficient dynamic resource scheduler for deep learning clusters. Proceedings of the European Conference on Computer Systems.

[6] Gu, J., Chowdhury, M., Shin, K. G., Zhu, Y., Jeon, M., Qian, J., et al. (2019). Tiresias: A GPU cluster manager for distributed deep learning. Proceedings of the USENIX Symposium on Networked Systems Design and Implementation.

[7] Mohan, J., Phanishayee, A., Raniwala, A., & Chidambaram, V. (2021). Analyzing and mitigating data stalls in DNN training. Proceedings of the VLDB Endowment.

[8] Jeon, M., Venkataraman, S., Phanishayee, A., Qian, J., Xiao, W., & Yang, F. (2019). Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. Proceedings of the USENIX Annual Technical Conference.

[9] Weng, Q., Xiao, W., Yu, Y., Wang, W., Wang, C., He, J., et al. (2022). MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. Proceedings of the USENIX Symposium on Networked Systems Design and Implementation.

[10] Mahajan, K., Balasubramanian, A., Singhvi, A., Venkataraman, S., Akella, A., Phanishayee, A., & Chawla, S. (2020). Themis: Fair and efficient GPU cluster scheduling. Proceedings of the USENIX Symposium on Networked Systems Design and Implementation.

[11] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., et al. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the USENIX Symposium on Networked Systems Design and Implementation.

[12] Or, A., Zhang, H., & Freedman, M. (2020). Resource elasticity in distributed deep learning. Proceedings of Machine Learning and Systems.

[13] Thinakaran, P., Gunasekaran, J. R., Sharma, B., Kandemir, M. T., & Das, C. R. (2019). Kube-Knots: Resource harvesting through dynamic container orchestration in GPU-based datacenters. Proceedings of the IEEE International Conference on Cluster Computing.

[14] Wu, Y., Ma, K., Yan, X., Liu, Z., & Cheng, J. (2021). Elastic deep learning in multi-tenant GPU cluster. arXiv preprint.

[15] Chen, Y., Ganapathi, A., Griffith, R., & Katz, R. (2011). The case for evaluating MapReduce performance using workload suites. Proceedings of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems

International Journal of Science, Technology and Convergence (IJSTC)
ISSN: 2134-986X

[16] Konda, P. R. (2026). Cloud-Native AI/ML Analytics Platform for Real-Time Enterprise Data Processing and Optimization. Synergia: A Journal of Multidisciplinary Innovation, 8(8). Retrieved from https://ijcdra.us/index.php/Synergia/article/view/72

[17] Sharma, M., Vangara, Y., Sharma, P., & Konda, P. R. (2025, June). NeuroNav: A Hybrid Deep Learning Framework for Sustainable Autonomous Indoor Robot Localization and Navigation. In International Conference on Sustainable Development through Machine Learning, AI and IoT (pp. 330-349). Cham: Springer Nature Switzerland.

[18] Konda, P. R. (2025). ADVANCED ENTERPRISE DATA ENGINEERING USING MACHINE LEARNING AND SCALABLE CLOUD ARCHITECTURES. Indonasian Journal of Advanced Research & Technology , 7(7). Retrieved from https://scholarlyarticle.vncinstitute.com/index.php/IJART/article/view/71

IJSTC