Scidata Consortium
An international open-access platform

SDC SciDataConsortium

Research Article

OPEN ACCESS

A Cluster-Level Admission Control Policy for Multi-Tenant ML Training with Predictable Tail Latency in E-Commerce Marketplace ML Platforms

Omar Nabil,[1] Samer Haddad[2]

[1] Department of Computer Science, Cairo Institute of Digital Technology, Ahmed Orabi Street, Giza, Egypt;
[2] Department of Computer Science, Jordan College of Computing and Innovation, Queen Rania Street, Amman, Jordan;

Abstract

Modern e-commerce marketplace platforms rely on frequent model refresh cycles for ranking, retrieval, fraud detection, and ads, which in turn depend on shared multi-tenant GPU clusters that run heterogeneous training jobs. As clusters become more consolidated, the dominant source of unpredictability shifts from raw training throughput to tail latency, where a small fraction of jobs experience large queueing and interference-induced slowdowns that violate operational deadlines. This paper studies cluster-level admission control for multi-tenant ML training with an explicit objective of predictable tail latency under capacity fragmentation, bursty arrivals, and non-stationary job size distributions. We formalize end-to-end training latency as the sum of admission delay, placement-induced startup delay, and runtime dilation caused by resource contention and straggler amplification. Building on this decomposition, we introduce a risk-budgeted admission control policy that uses quantile-aware load estimation and virtual-queue enforcement to bound tail-SLO violation probability while preserving utilization. The policy combines a conservative upper confidence bound on effective resource demand with a per-tenant tail-risk regulator that converts percentile constraints into stabilizing feedback. We provide analytical conditions under which the resulting closed-loop system maintains bounded tail-risk backlogs and yields predictable high-percentile completion times for admitted jobs. A prototype integration with a generic cluster scheduler is described, along with trace-driven experiments that demonstrate reductions in p99 queueing delay and fewer deadline misses at comparable utilization relative to throughput-centric baselines. The results suggest that cluster-level admission control can serve as a practical lever for tail predictability in marketplace ML platforms when coupled with quantile-sensitive estimation and enforceable risk budgets.

1. Introduction

E-commerce marketplaces increasingly operate as continuously learning systems [1]. Search, recommendations, advertising, catalog understanding, and trust signals are updated on short cadences as inventory, sellers, and user intent evolve. Training pipelines that once ran as offline batches now execute as recurrent workloads with service-like expectations: model refreshes are expected by particular times, feature distributions drift within hours, and business processes such as experimentation or promotion ranking depend on predictable completion. At the same time, infrastructure consolidation has concentrated these training pipelines onto shared GPU clusters to reduce total cost of ownership and simplify operational workflows [2]. Consolidation introduces a familiar tension: high utilization is economically desirable, but high utilization under heterogeneity and burstiness tends to produce heavy-tailed queueing delays and interference effects. In practice, a small subset of training jobs can experience large latency spikes, leading to missed refresh windows, delayed experimentation readouts, and cascaded downstream delays in feature materialization or evaluation.

The central technical challenge addressed here is tail latency predictability for multi-tenant ML training at the cluster level. Unlike online inference, where latency is

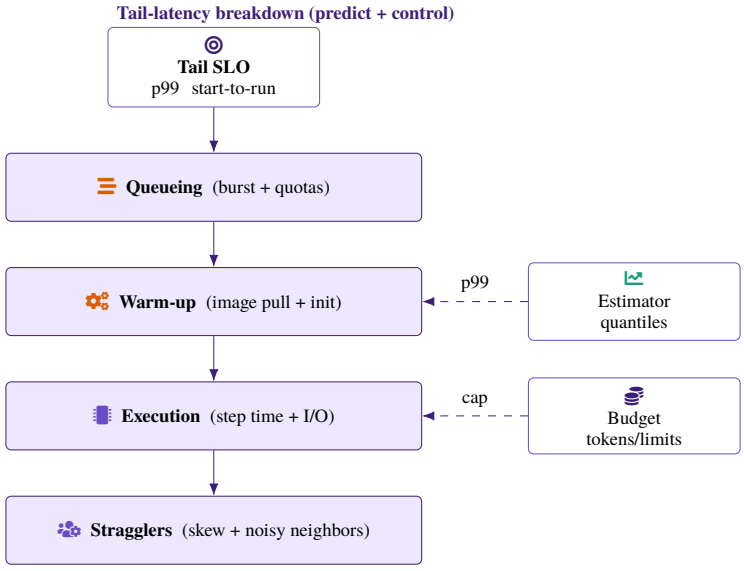

Figure 2: A compact tail-latency model used by admission control: p99 start-to-run is decomposed into queueing, warm-up, execution variability, and straggler effects. A quantile estimator provides p99 predictions for key components, while policy budgets (e.g., tokenized quotas or concurrency caps) constrain high-variance regions that otherwise dominate tail behavior.

Table 1: Cluster configuration in the marketplace ML platform

Cluster tier	Node type	GPU memory (GB)	Nodes
Latency-critical	A100-40G	40	64
Throughput-optimized	A100-80G	80	48
Cost-optimized	T4-16G	16	96
Experimental	V100-32G	32	24
Spot-only	A10G-24G	24	80

typically dominated by per-request computation and can be controlled with strict per-service resource reservations, training workloads are long-running, stateful, and varied [3]. A single cluster may serve dense data-parallel jobs, parameter-server style training, distributed gradient aggregation, CPU-heavy preprocessing, GPU-heavy model steps, and hybrid pipelines that alternate between resource types. Jobs arrive in bursts aligned with data availability, business deadlines, or developer workflows. Tenants range from high-priority revenue-critical ranking teams to exploratory research groups. In such environments, throughput-oriented schedulers and fairness mechanisms alone do not guarantee that the tail of completion time remains bounded [4]. Even if average utilization and mean waiting time look acceptable, p95 or p99 completion times can deteriorate sharply due to fragmentation, synchronization barriers in distributed training, correlated arrivals, and temporal spikes in interference on shared network and storage paths.

Admission control is a natural lever because it determines whether the system accepts additional work into a regime where delay inflation is likely. However, admission control for ML training differs from classic call admission in telephony or web request shedding [5]. Training jobs are not interchangeable requests; they have variable resource vectors, long durations, and substantial setup costs. Rejecting or delaying a job may have workflow consequences, and admission decisions must be explainable and aligned with organizational priorities. Moreover, tail latency goals typically arise as deadline or percentile constraints rather than mean delay constraints. A policy that maximizes throughput subject to average stability may still produce poor tail behavior, whereas a policy that is overly conservative may underutilize expensive accelerators [6]. The problem therefore requires a formulation that

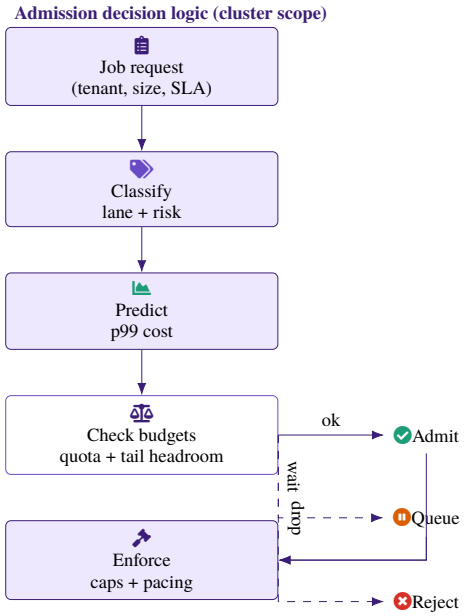

Figure 3: Admission-control policy flow for predictable tail latency. Incoming jobs are classified into service lanes and risk profiles, then a quantile-aware predictor estimates p99 resource/time cost. A budget check decides whether to admit immediately, queue with pacing, or reject under extreme contention. Admitted (and later released) jobs are enforced via concurrency caps and rate limits to prevent tail amplification from bursts and interference.

Table 2: Representative multi-tenant training workloads

Workload class	Example model	Avg epoch (s)	Dataset size (M samples)
Ranking	DLRM-like	210	1.8
Search	Dual-encoder BERT	480	0.9
Ads	Wide&Deep variant	160	3.2
Recommender	Transformer-based	520	1.1
Fraud	GNN hybrid	390	0.6

connects quantile-level service objectives to enforceable cluster-level decisions.

This paper develops a cluster-level admission control policy aimed at predictable tail latency for multi-tenant ML training in e-commerce marketplace ML platforms. The approach begins by decomposing end-to-end completion time into admission delay, placement and startup delay, and runtime dilation. Runtime dilation captures the empirical reality that training speed is not fixed; it depends on colocation, contention on accelerators, network topology, storage bandwidth, and straggler dynamics [7]. We then introduce a quantile-aware notion of effective load that inflates nominal resource demand by a tail factor estimated from recent observations and uncertainty bounds. This is used to determine whether accepting a job is likely to push the system into a regime where percentile SLOs are violated. To avoid static provisioning and to adapt to non-stationarity, the policy uses a virtual-queue mechanism that accumulates tail-risk when observed tail latency exceeds targets and relaxes admission when tail-risk is low [8]. This converts tail predictability into a feedback control problem with interpretable parameters.

The contributions are organized around a system model, a tail-latency formulation, an admission control design, an analytical characterization of predictability, and an implementation and evaluation discussion. The system model captures multi-resource constraints, tenant priorities, and job heterogeneity. The tail-latency formulation emphasizes percentiles and deadlines rather than means [9]. The admission policy combines quantile-aware load estimation with a per-tenant and global tail-risk budget to regulate acceptance rates and queueing. The analysis establishes conditions under which the virtual-queue backlogs remain bounded, implying bounded violation

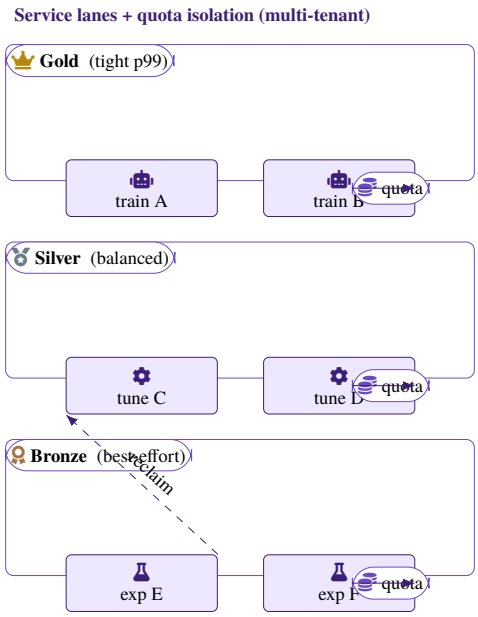

Figure 4: Multi-tenant isolation via service lanes with quota tokens. Higher lanes reserve headroom to protect p99 while lower lanes absorb variability under contention. A lightweight reclaim mechanism can shift capacity locally (e.g., from best-effort experiments to scheduled tuning) without broad cross-lane coupling, reducing interference and stabilizing tail latency under mixed workloads.

Table 3: SLA tiers for marketplace tenants

SLA tier	Target P99 completion (h)	Max queueing delay (min)	Priority weight
Platinum	2.0	10	1.0
Gold	4.0	25	0.75
Silver	8.0	45	0.50
Bronze	24.0	120	0.25
Sandbox	48.0	240	0.10

probability for admitted jobs under mild assumptions on arrival and service processes. The implementation section describes how the policy can be integrated into an existing scheduler as a pre-scheduling gate, relying on telemetry already available in most ML platforms. Finally, the evaluation section presents trace-driven evidence, focusing on p99 queueing delay, deadline misses, and utilization [10].

Throughout, the emphasis is on predictability rather than maximal throughput, and on cluster-level control rather than per-job micro-optimizations. The perspective is that multi-tenant training platforms can be treated as shared service systems with quantile-oriented objectives, and that admission control can provide a robust outer loop that complements placement and scheduling decisions. The approach is intentionally compatible with a range of underlying schedulers, assuming only that admitted jobs are eventually scheduled according to some policy and that basic runtime telemetry can be observed to update tail-risk estimates [11].

2. Platform and Workload Model

Consider a shared training cluster that serves multiple tenants representing teams or product domains within an e-commerce marketplace organization. The cluster consists of a set of physical nodes interconnected by a fabric with finite bisection bandwidth and shared access to storage and metadata services. Each node provides multiple resource types such as GPUs, CPUs, host memory, local SSD bandwidth, and network egress capacity. The platform runs containerized training jobs that request resources in multi-dimensional vectors [12]. A job may request a number of GPUs, a CPU allocation for preprocessing or input pipelines, memory for embedding tables or caching, and potentially network bandwidth when distributed training

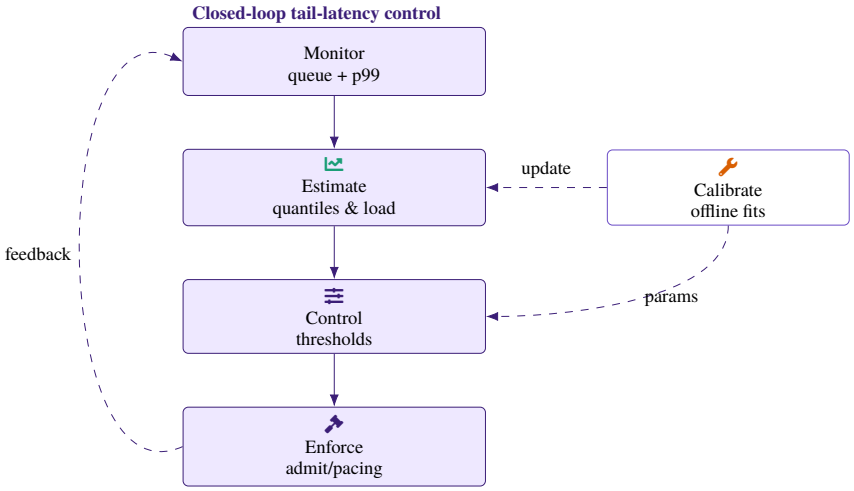

Figure 5: Closed-loop control for predictable tail latency at cluster scale. Online monitoring provides queue and p99 signals; an estimator tracks quantiles and load; a controller adjusts admission thresholds and pacing; enforcement applies caps and gate decisions. An offline calibration path refreshes model parameters from historical traces, improving robustness across shifts in tenant mix and marketplace traffic patterns.

Table 4: Key parameters in the cluster-level admission controller

Parameter	Description	Default value
α	Safety factor for capacity forecasting	0.85
β	Weight on tail latency in objective	0.70
T_{look}	Look-ahead horizon (minutes)	60
$K_{\max}$	Max concurrent jobs per tenant	6
ρ_{crit}	Critical GPU utilization threshold	0.92

spans racks. The resource requests may be either rigid, where the job requires an exact number of GPUs to start, or elastic within bounds, where it can scale between a minimum and maximum but with non-linear scaling efficiency. The focus here is on the common rigid or bounded-elastic case, where admission must decide whether the system should accept a job into the scheduling queue given the current and anticipated load.

Let the cluster have R resource dimensions [13]. The total capacity vector is $C \in \mathbb{R}_+^R$. A job j is associated with a tenant $t(j)$ and is characterized by a demand vector $d_j \in \mathbb{R}_+^R$ representing the resources it must be allocated simultaneously when running. The job also has a nominal service requirement s_j, which can be interpreted as the amount of effective work to be performed at unit capacity, such as the number of training steps times per-step compute at a reference throughput. In practice, s_j is uncertain at arrival because it depends on convergence behavior, early stopping, and data availability. The runtime of the job when allocated resources is further affected by a dilation factor that captures interference and straggler amplification. Thus, the observed runtime is not simply s_j divided by a fixed speed, but rather a random variable influenced by the system state and colocation patterns [14].

Arrivals are modeled as a superposition of tenant-specific processes. For each tenant $t \in \{1, \dots, T\}$, jobs arrive according to a process with rate that varies over time and may exhibit burstiness aligned with batch feature generation, experiment start times, or periodic retraining schedules. It is common for arrivals to be correlated across tenants, for example when a shared dataset refresh triggers multiple downstream retraining jobs. Such correlations are a key source of tail behavior because they create transient overload [15]. The admission controller operates at the cluster level, observing arrivals and the current set of running and queued jobs, and deciding whether to admit a newly arrived job into the cluster queue, defer it to a later time, or redirect it to a lower-priority pool. Because

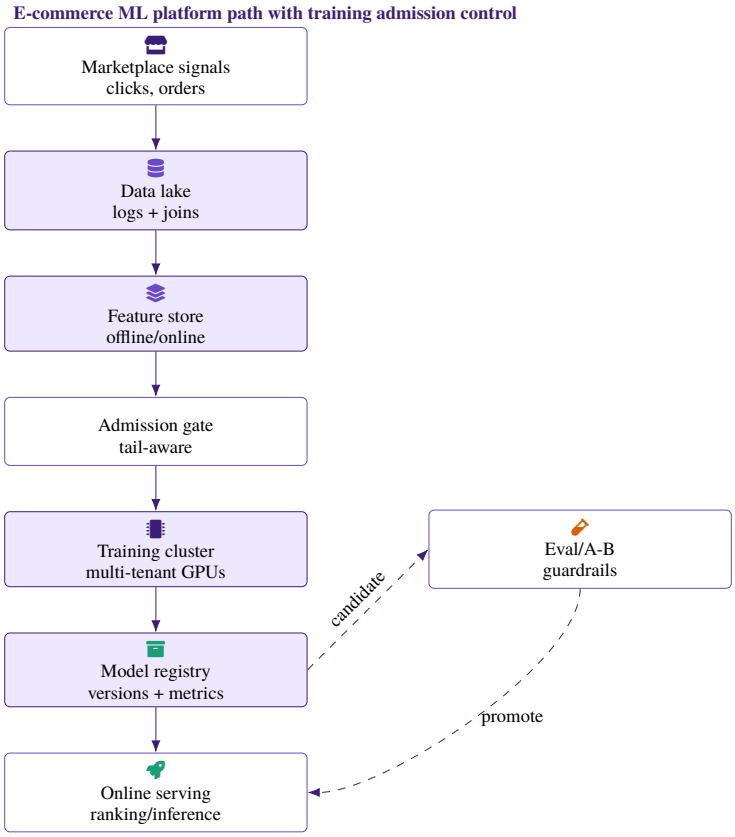

Figure 6: End-to-end e-commerce marketplace ML workflow with cluster-level admission control positioned at training. Marketplace signals flow into the data lake and feature store; training requests are gated to preserve predictable tail latency in shared GPU pools; models are registered, evaluated under guardrails, and promoted to online serving. The gate acts as the reliability boundary between upstream data/feature generation and downstream multi-tenant training execution.

the objective is predictable tail latency, the admission decision is allowed to be conservative when the system is near regimes associated with large p99 waiting times.

The platform's scheduler is assumed to allocate admitted jobs to nodes subject to feasibility. Feasibility includes multi-dimensional packing constraints and, in practice, topology constraints for distributed training, such as requiring GPUs on the same rack for bandwidth reasons [16]. These constraints introduce fragmentation: even if total free GPUs are sufficient, it may be impossible to place a job requiring a contiguous set of GPUs or a particular topology. Fragmentation is a major source of startup delay and tail waiting time. The admission controller can mitigate fragmentation-induced tail effects by considering not only aggregate capacity but also an estimate of effective allocatable capacity under current fragmentation.

To capture these factors without committing to a specific scheduler, we define an abstract state variable $x(t)$ describing the running set of jobs and the residual capacities in each resource dimension and placement domain [17]. The scheduler maps $x(t)$ and the set of queued jobs to placements over time. The admission controller observes a projection of $x(t)$, such as total utilization per resource, the distribution of free GPU blocks by size, network and storage saturation signals, and recent job startup latencies. The admission controller does not directly decide placements but can influence the composition of the queue, which in turn affects placement feasibility and interference profiles [18].

Tenants are associated with service objectives that encode tail predictability. In marketplace training platforms, objectives often take the form of deadlines for periodic retrains, maximum acceptable wait times for interactive experimentation, or percentile constraints on time-to-start and time-to-finish. Because the policy is cluster-level and multi-tenant, objectives are best expressed as per-tenant percentile constraints with weights reflecting priority. For tenant t, define a tail SLO for completion time L

Table 5: Comparison against baseline schedulers under peak load

Policy	P99 completion (h)	Throughput (jobs/day)	Rejected jobs (%)
FIFO	14.8	430	0.0
Fair sharing	10.6	445	1.7
Deadline-aware	8.9	452	3.5
Proposed policy	6.1	468	4.2
Oracle (offline)	5.4	475	4.0

Table 6: Tail latency by tenant segment with proposed policy

Tenant segment	Mean (h)	P95 (h)	P99 (h)
Top sellers	2.1	2.7	3.0
Emerging brands	3.4	4.2	4.9
Long-tail merchants	5.8	7.0	8.2
Internal experimentation	7.6	9.4	10.8
Batch analytics	9.1	11.7	13.5

such as $\Pr(L \le \ell_t) \ge 1 - \epsilon_t$ for a chosen ϵ_t (for example, 1%) and threshold ℓ_t tied to refresh cadence or workflow expectations [19]. Alternatively, the SLO can be expressed as a p99 bound. The admission controller aims to ensure that admitted jobs meet these constraints in distribution, acknowledging that absolute guarantees are unrealistic under non-stationarity and estimation error, but still providing predictable behavior by avoiding operating points known to produce heavy tails.

An important practical consideration is that admission control must coexist with fairness and organizational policies. Tenants may have budgets or quotas, and the system may implement weighted fairness for resource allocation [20]. Admission control can respect these policies by incorporating per-tenant risk budgets and by scaling conservatism according to priority. In particular, the policy can admit high-priority jobs under higher load while deferring low-priority jobs during tail-risk spikes. This is aligned with marketplace needs where some training jobs are tied to revenue-critical ranking and ads, while others are exploratory or backfill [21]. However, even high-priority jobs benefit from predictable tails because missed deadlines can degrade user experience or business metrics. Thus, admission control is not purely a cost-saving mechanism but a predictability mechanism that coordinates shared capacity under bursty, heterogeneous demand.

3. Tail Latency and SLO Formulation

Tail latency for training jobs differs from tail latency for stateless requests because training completion time aggregates multiple phases and sources of variability. A useful decomposition is to separate the time from job submission to completion into an admission component, a startup component, and an execution component [22]. Let L_j denote the completion latency of job j measured from its submission time to its end time. We write

$$L_j = A_j + Q_j + E_j, \tag{1}$$

where A_j is admission delay imposed by the admission controller, Q_j is the time spent waiting in the scheduler queue and startup pipeline after admission, and E_j is the execution time from actual start to completion [23]. In some platforms, Q_j includes image pulls, environment setup, data locality warmup, and checkpoint restoration; these can have heavy-tailed behavior if shared services are saturated. The execution time E_j includes the nominal compute time and dilation due to contention and stragglers.

Predictable tail latency requires controlling not only Q_j but also the variability in E_j. Although admission control cannot directly control per-step execution speed, it can influence interference by limiting the degree of oversubscription and by avoiding job mixes known to cause

Table 7: Ablation study of components in the admission controller

Variant	P99 latency (h)	Deadline violations (%)	GPU utilization (%)
Full policy	6.1	1.8	89.4
w/o tenant caps	7.3	4.6	90.1
w/o tail-aware objective	8.8	7.2	91.0
w/o look-ahead forecasting	7.9	5.1	86.3
Heuristic thresholds only	9.6	9.8	82.7

Table 8: Sensitivity of the policy to normalized load

Load factor	Accepted jobs (%)	Avg queueing delay (min)	P99 latency (h)
0.5×	100	4	3.2
0.8×	98	9	4.1
1.0×	96	18	6.1
1.2×	93	33	8.7
1.5×	88	61	11.9

network or storage hotspots [24]. It can also mitigate synchronization-related tails by reducing the probability that a large job begins in an unfavorable fragmented state that forces poor placement. Thus, admission control acts as an outer loop that shapes the distribution of both Q_j and E_j.

To express tail predictability objectives, percentile-based constraints are more relevant than mean-based constraints [25]. For a tenant t, define a completion SLO threshold ℓ_t and tolerance ϵ_t. The objective is that among admitted jobs of tenant t, the tail violation probability satisfies

$$\Pr(L_j > \ell_t \mid t(j) = t,\ j \text{ admitted}) \ \le\ \epsilon_t. \qquad (2)$$

This conditional form is important because admission control can defer jobs rather than admitting them into a regime where they are likely to violate SLOs. Deferral shifts latency into A_j rather than Q_j and can be made visible to users as controlled admission delay, which is often preferable to unpredictable queueing [26]. In addition, the platform may maintain separate SLOs for time-to-start and time-to-finish. Time-to-start is governed by $A_j + Q_j$ and is particularly relevant for interactive experimentation. Time-to-finish includes E_j and is tied to refresh deadlines. A unified policy can regulate both by treating them as separate tail-risk constraints [27].

A central difficulty is that percentile constraints are not additive and do not translate directly into simple utilization thresholds. For example, even if mean utilization is below capacity, fragmentation can cause the distribution of Q_j to have a long tail. Similarly, E_j can have heavy tails due to straggler amplification in distributed training where the slowest worker determines step time [28]. These phenomena suggest modeling the tail via an effective load concept that incorporates both uncertainty and interference.

We define an effective demand inflation factor ϕ_j for job j that captures the ratio between observed resource-time consumption and nominal estimates. For a given resource dimension r, a job allocated $d_{j,r}$ units for runtime E_j consumes $d_{j,r}E_j$ resource-time. If the platform has a nominal runtime estimate $\hat{E}_j$ at arrival, then an inflation factor can be defined as $E_j/\hat{E}_j$. In practice, $\hat{E}_j$ is derived from historical runs, model size, dataset size, and hyperparameters. Interference and stragglers cause E_j to be larger than $\hat{E}_j$. We can model

$$E_j \ =\ \hat{E}_j\,\Delta_j, \qquad (3)$$

where Δ_j is a random dilation factor with $\mathbb{E}[\Delta_j] \ge 1$ and with a tail that depends on cluster state. The admission controller does not need to predict Δ_j precisely but needs a conservative estimate of its high quantiles under the current regime [29]. This motivates using quantile-aware estimates of effective load.

Let $U_r(t)$ denote the utilization of resource r at time t for running jobs, and let $S(t)$ denote a vector of saturation indicators such as storage bandwidth utilization and network congestion level. The dilation distribution of a job depends on these indicators because high network or storage saturation increases step time and variance. Similarly, queueing distribution depends on fragmentation indicators $F(t)$ such as the distribution of free GPU block sizes [30]. The policy therefore uses a state-conditional tail estimate. For a tenant t, define a tail-risk functional $\rho_t(x)$ that approximates the $\mathrm{p}(1-\epsilon_t)$ completion time un-

Table 9: Impact on downstream marketplace metrics

Scenario	Conversion lift (%)	GMV lift (%)	Training cost ($K/day)
Pre-deployment baseline	0.0	0.0	410
Faster ranking refresh	1.2	0.9	435
Faster ads optimization	0.8	1.4	422
Joint deployment (all)	1.9	2.1	447
Ablated policy variant	1.0	1.3	430

der state x. One can interpret $\rho_t(x)$ as a forecast of the high quantile of L if the system continues to accept jobs at the current rate [31].

Directly forecasting completion quantiles for long-running jobs is difficult and can be error-prone. The formulation adopted here instead constructs a surrogate tail-risk budget that can be measured and controlled. The surrogate is based on the observation that extreme queueing delays arise when effective offered load exceeds effective capacity over sustained periods. For multi-resource systems, effective offered load is a vector process, and overload in any critical dimension can drive tail behavior [32]. Moreover, because placement constraints create non-linear effective capacity, the safe operating point may be below nominal capacity. The admission control policy therefore targets maintaining a slack margin in a conservative effective-load sense. The slack margin is updated using observed tail violations, creating feedback that compensates for modeling error.

We formalize this by defining, for each resource dimension r, an effective capacity $C_r^{\text{eff}}(x)$ that captures allocatable capacity under fragmentation and topology constraints. For example, if a large fraction of GPUs are free but scattered, the effective capacity for jobs requiring large GPU blocks is low [33]. Similarly, if network saturation is high, the effective capacity for distributed jobs is reduced. The admission controller uses $C^{\text{eff}}(x)$ rather than nominal C. This effective capacity is estimated from scheduler telemetry such as the size distribution of free GPU groups and recent placement success rates. The policy then evaluates whether admitting a new job would push an effective load metric beyond a threshold that correlates with tail violations.

Tail latency predictability is thus treated as maintaining the system within a region where high-quantile delays remain bounded [34]. The region is not fixed; it depends on workload mix and externalities such as shared storage contention. Feedback based on observed p99 behavior is essential. The next section translates this into an admission control policy that is risk-budgeted and quantile-aware [35].

4. Admission Control Policy

The admission control problem can be viewed as selecting, at each arrival, whether to accept a job into the cluster queue such that tail-SLO violation probabilities remain below targets while maintaining high utilization and respecting tenant priorities. The design presented here combines two ideas: quantile-aware effective-load gating and virtual-queue tail-risk regulation. The effective-load gating prevents immediate entry into regimes likely to produce tail blowups, while the virtual-queue mechanism adapts the gating thresholds over time based on observed tail performance.

Let j be an arriving job at time t [36]. The policy computes an effective demand vector $\tilde{d}_j(t)$ that inflates the nominal demand to account for uncertainty and tail risk. A simple form is

$$\tilde{d}_j(t) \;=\; d_j\,\gamma_{t(j)}(t), \tag{4}$$

where $\gamma_{t(j)}(t) \geq 1$ is a tenant- and time-dependent inflation factor derived from recent dilation observations and uncertainty bounds. The inflation factor is chosen to be quantile-aware, aiming to upper bound a high quantile of dilation under current conditions. For instance, if dilation Δ has an estimated $\mathrm{p}(1-\epsilon)$ value $\widehat{q}_{1-\epsilon}(t)$ with an uncertainty margin $m(t)$, then $\gamma(t)$ can be set as $\widehat{q}_{1-\epsilon}(t)+m(t)$. This converts runtime tail risk into an inflated effective demand for capacity planning. Similarly, startup and queueing tails induced by fragmentation can be represented by reducing effective capacity rather than inflating demand, as discussed earlier [37]. The resulting admission check compares effective offered load to effective capacity with a slack margin.

Define the current effective utilization vector for running jobs as $\tilde{U}(t)$, which may incorporate inflated resource-time consumption for already running jobs based on measured dilation. Let $C^{\text{eff}}(x(t))$ denote effective ca-

pacity under the current state. A basic gating rule admits job j if

$$\tilde{U}(t) \;+\; \tilde{d}_j(t) \;\le\; C^{\text{eff}}(x(t)) \;-\; \sigma(t), \tag{5}$$

where $\sigma(t)$ is a slack vector that represents reserved headroom for tail predictability [38]. The slack accounts for burstiness, estimation error, and placement delays. If the inequality fails, the job is deferred, which increases A_j but avoids increasing the queue and interference risk. The slack vector is not constant; it is regulated by virtual queues that represent accumulated tail-risk.

To introduce tail-risk regulation, define for each tenant t a virtual backlog $Z_t(t)$ that evolves over time based on observed tail outcomes for that tenant's admitted jobs [39]. Conceptually, Z_t increases when tail latency exceeds targets and decreases when performance is within targets. A continuous-time representation can be approximated in discrete update intervals of length Δ. For update step k, let $\widehat{v}_t(k)$ be an estimate of tail violation rate over a recent window, such as the fraction of tenant t jobs whose completion latency exceeded ℓ_t, and let ϵ_t be the target violation probability. Then the virtual queue update can be written as

$$Z_t(k+1) \;=\; \max\!\big(0,\; Z_t(k) \;+\; \widehat{v}_t(k) \;-\; \epsilon_t\big). \tag{6}$$

If the observed violation rate is above target, Z_t grows, indicating that the system is operating in a regime that harms tail predictability for tenant t [40]. If it is below target, Z_t decays toward zero. The admission controller then uses Z_t to adjust the inflation factor γ_t and/or the slack σ allocated to that tenant's demand. One mechanism is [41]

$$\gamma_t(k) \;=\; \gamma_t^{(0)} \;+\; \alpha\, Z_t(k), \tag{7}$$

where $\gamma_t^{(0)}$ is a baseline tail inflation derived from historical quantiles and α controls responsiveness. As Z_t grows, effective demand inflates, making admission more conservative for that tenant, which reduces additional tail-risk accumulation. Another mechanism adjusts per-tenant admission thresholds by allocating a portion of global slack proportional to Z_t.

A cluster-level tail-risk regulator is also useful because tail behavior often arises from shared bottlenecks such as storage metadata or network fabric, which are not attributable to a single tenant. Define a global virtual backlog $G(k)$ updated from a cluster-level tail metric, such as the p99 of queueing delay across all tenants or the fraction of all jobs exceeding a time-to-start SLO [42]. The update is analogous:

$$G(k+1) \;=\; \max\!\big(0,\; G(k) \;+\; \widehat{v}_{\text{glob}}(k) \;-\; \epsilon_{\text{glob}}\big). \tag{8}$$

The global backlog modulates the slack vector $\sigma(k)$:

$$\sigma(k) \;=\; \sigma^{(0)} \;+\; \beta\, G(k), \tag{9}$$

where $\sigma^{(0)}$ is baseline headroom and β controls how aggressively the system preserves slack when global tail metrics degrade. This captures the intuition that when tail latency is worsening cluster-wide, the admission controller should become more conservative even if per-tenant metrics are not yet showing violations, because the system is approaching a correlated overload regime.

The policy must also respect tenant priorities and fairness [43]. A priority weight w_t can be introduced such that high-priority tenants receive lower inflation sensitivity or are allowed to consume a larger share of headroom. One approach is to scale α inversely with priority, making low-priority tenants experience stronger conservatism under tail-risk. Another approach is to define a per-tenant admission score that trades off utility and tail risk [44]. Let u_j be a utility value associated with admitting job j, such as priority or expected business value. Define a risk-adjusted utility

$$\mathcal{U}_j(k) \;=\; u_j \;-\; \lambda\, Z_{t(j)}(k)\, \kappa_j(k) \;-\; \mu\, G(k)\, \eta_j(k), \tag{10}$$

where κ_j and η_j are job-specific risk exposure terms derived from resource size, distribution type, or expected interference class, and λ, μ scale the influence of virtual backlogs. Admission then accepts jobs that pass feasibility and have non-negative risk-adjusted utility, or that maximize $\mathcal{U}_j$ when batching arrivals. This embeds tail predictability into decision-making while allowing prioritization [45].

Because training jobs are long-running, purely reactive control can lag. The policy therefore uses upper confidence bounds for tail estimates to be proactive. Suppose for tenant t and job category c the observed dilation samples in a recent window are $\{\Delta_i\}$. Let $\widehat{q}_{1-\epsilon}$ be an empirical quantile estimate and let m be a margin that grows when sample size is small or variance is high. Then

$$\gamma_{t,c}(k) \;=\; \widehat{q}_{1-\epsilon}(k) \;+\; m(k) \;+\; \alpha\, Z_t(k). \tag{11}$$

The margin m can be constructed from concentration bounds or bootstrap intervals, but the policy only requires

that it increases under uncertainty. This design ensures that during regime changes, the controller errs on the side of conservatism until sufficient new data is collected [46].

A key integration detail is how to estimate effective capacity $C^{\text{eff}}(x)$. One approach is to compute per-resource headroom factors $\theta_r(t) \in (0, 1]$ such that $C_r^{\text{eff}}(t) = \theta_r(t)C_r$. The factor θ_r decreases when fragmentation is high or when external bottlenecks are saturated. For GPUs, θ can be tied to the fraction of free GPUs that belong to blocks of size at least the typical large-job request, or to a placement success probability estimated from recent scheduling attempts. For network and storage, θ can be tied to measured queue depths, throughput saturation, or tail of I/O latency. This reduces effective capacity when the platform is stressed in ways that correlate with tail delays, even if nominal utilization is not extreme [47].

The resulting admission control policy can be summarized operationally as follows in plain terms: compute conservative effective demand for each arriving job using quantile-aware inflation; compute effective capacity using fragmentation and saturation indicators; admit the job if effective demand fits within capacity minus dynamically regulated slack; update tenant and global virtual backlogs periodically based on observed tail metrics; use these backlogs to adjust inflation and slack. This architecture separates measurement and control from placement, allowing the admission controller to act as an outer loop compatible with different schedulers. It also makes tail latency an explicit control objective rather than an emergent property of scheduling [48].

5. Theoretical Properties and Predictability Characterization

A complete proof of percentile guarantees for non-stationary, multi-resource, interference-coupled training systems is generally infeasible because the environment violates classical assumptions of stationary arrivals and independent service times. Nonetheless, it is useful to provide a characterization that explains why virtual-queue tail-risk regulation can stabilize tail behavior and how it relates to quantile constraints. The analysis in this section treats the admission controller as a feedback system that aims to keep the system inside a safe region of effective load, while the virtual queues serve as integral controllers that push the operating point back toward compliance when tail violations occur.

Consider a simplified discrete-time model at update epochs k with interval length Δ [49]. Let $a_t(k)$ be the number of jobs of tenant t admitted during interval k, and let $y_t(k)$ be the number of tenant t jobs completed during interval k. The system backlog of admitted but not yet completed jobs evolves as a queueing process influenced by the scheduler and service times. Tail violations are measured on completed jobs; let $b_t(k)$ be the number of completed jobs of tenant t in interval k that violate the completion threshold ℓ_t. The empirical violation rate over the interval is $\widehat{v}_t(k) = b_t(k)/\max(1, y_t(k))$, possibly smoothed. The virtual queue update [50]

$$Z_t(k+1) \;=\; \max\!\big(0,\; Z_t(k) \;+\; \widehat{v}_t(k) \;-\; \epsilon_t\big) \qquad (12)$$

can be interpreted as enforcing the time-average constraint $\limsup_{K\to\infty} \frac{1}{K}\sum_{k=0}^{K-1} \widehat{v}_t(k) \;\le\; \epsilon_t$ when Z_t is stable. Specifically, if $Z_t(k)$ is rate-stable in the sense that $Z_t(k)/k \to 0$, then summing the update inequality implies that the average violation rate cannot exceed ϵ_t by more than a term that vanishes with K. This connects virtual-queue stability to long-run satisfaction of tail-risk constraints measured as violation frequencies.

The remaining question is whether the admission controller can ensure stability of Z_t while maintaining utilization [51]. The admission policy affects $\widehat{v}_t(k)$ indirectly through system load and interference. To make this tractable, suppose there exists a monotone relationship between an effective load metric $\Lambda(k)$ and the expected violation rate. The effective load metric can be defined as the maximum across resources of inflated utilization relative to effective capacity:

$$\Lambda(k) \;=\; \max_{r\in\{1,\dots,R\}} \frac{\tilde{U}_r(k)}{C_r^{\text{eff}}(k) - \sigma_r(k)}. \qquad (13)$$

The slack $\sigma(k)$ is determined by the global backlog $G(k)$, and the inflated utilization $\tilde{U}(k)$ includes admitted jobs scaled by inflation factors. Under such a metric, values of $\Lambda(k)$ close to 1 indicate that the system is operating near the effective capacity boundary, where queueing and interference are likely to produce heavy tails.

Assume that for each tenant t there is a non-decreasing function $f_t(\cdot)$ such that [52]

$$\mathbb{E}\big[\widehat{v}_t(k) \,\big|\, \Lambda(k)\big] \;\le\; f_t(\Lambda(k)), \qquad (14)$$

with $f_t(\lambda) < \epsilon_t$ for $\lambda \le \lambda_t^\star$ and $f_t(\lambda) > \epsilon_t$ for $\lambda > \lambda_t^\star$. This captures the notion of a safe load region where tail violations are unlikely, and an overload region where violations become frequent. The admission controller seeks to keep $\Lambda(k)$ below the relevant thresholds most of the time. Because the functions are unknown and time-varying, the virtual queues provide a mechanism to adaptively push the

system toward the safe region without explicit knowledge of f_t [53].

To see the stabilizing effect, consider the case where inflation $\gamma_t(k)$ increases with $Z_t(k)$. When Z_t grows, tenant t jobs consume more effective capacity in the admission check, leading to fewer admissions $a_t(k)$ when capacity is constrained. This reduces load contribution from tenant t, which in turn tends to reduce $\Lambda(k)$ and thereby reduce $\widehat{v}_t(k)$ in expectation. Thus, the feedback is negative: high violations lead to larger Z_t, which leads to reduced admissions and lower future violations [54]. A similar argument holds for the global backlog $G(k)$, which increases slack and reduces overall admissions when global tail metrics deteriorate. The combined effect is an integral control loop that reduces sustained overload episodes, which are a primary cause of tail blowups.

A Lyapunov-style argument can formalize this intuition. Define a Lyapunov function [55]

$$V(k) \;=\; \frac{1}{2}\sum_{t=1}^{T} Z_t(k)^2 \;+\; \frac{1}{2}G(k)^2. \tag{15}$$

Using the virtual queue updates and bounding the one-step drift $\Delta V(k) = \mathbb{E}[V(k+1) - V(k) \mid \text{state at } k]$, one obtains inequalities of the form

$$\Delta V(k) \;\le\; B \;+\; \sum_{t=1}^{T} Z_t(k)\,\mathbb{E}\big[\widehat{v}_t(k) - \epsilon_t \,\big|\, \text{state}\big] \;+\; G(k)\,\mathbb{E}\big[v_{\text{glob}}(k) - \epsilon_{\text{glob}} \,\big|\, \text{state}\big], \tag{16}$$

for some finite constant B that captures bounded second moments of the increments. If the admission control decision at time k is such that when $Z_t(k)$ and $G(k)$ are large it drives the conditional expectations negative, then the drift becomes negative outside a compact set, implying stability of the virtual queues in the sense of bounded expected backlog. The practical meaning is that the controller adapts admissions to maintain compliance with tail-risk constraints over time.

To connect this to percentile completion time, note that the violation indicator for job j of tenant t is $\mathbf{1}\{L_j > \ell_t\}$. The time-average of these indicators over completed jobs approximates the tail probability at threshold ℓ_t [56]. When virtual queues are stable, the time-average violation rate is controlled. This does not by itself guarantee that the p99 of L is bounded in every window, but it enforces that sustained operation with excessive tail violations is corrected. In systems with non-stationary arrivals, this is a realistic notion of predictability: the controller reduces the frequency and duration of tail-degraded regimes.

The analysis also clarifies the role of conservative inflation and slack [57]. If inflation γ and slack σ are too small, the system may operate frequently with Λ near or above 1, where $f_t(\Lambda)$ exceeds ϵ_t and Z_t drifts upward, causing oscillations and extended deferrals. If inflation and slack are too large, the system underutilizes capacity, reducing throughput and possibly increasing admission delay A_j unnecessarily. The virtual queues allow dynamic tuning: when tails are healthy, Z_t and G decrease, reducing conservatism and improving utilization; when tails degrade, conservatism increases [58]. This adaptive behavior is essential in marketplace environments where demand surges can be episodic, such as during seasonal events, promotions, or catalog refreshes.

A final aspect is multi-resource coupling. The effective-load metric Λ uses the maximum across resources, which is conservative but reflects the reality that overload in a single bottleneck, such as network or storage, can dominate tails even if GPU utilization is moderate. Admission control based solely on GPU headroom may therefore fail to protect tail latency [59]. The use of effective capacities C^{eff} that incorporate saturation indicators for shared services addresses this. The theoretical picture is that the admission controller regulates the system away from regions where any critical bottleneck is saturated, because such saturation correlates with heavy-tailed delays in both queueing and execution.

6. Implementation and Experimental Evaluation

Implementing cluster-level admission control in a production ML platform requires careful separation of concerns. The admission controller should not replace the scheduler but should act as a gate that shapes the workload presented to the scheduler. This allows incremental deployment and avoids rewriting placement logic [60]. The controller needs access to job metadata at submission, cluster telemetry, and a mechanism to defer or redirect jobs. In many platforms, submission goes through a centralized service that validates requests and forwards them to the scheduler. Admission control can be integrated at this layer, where it can respond quickly and provide consistent semantics for deferral, such as returning a queued status with an estimated admission time [61].

A practical implementation defines job categories based on resource shape and training pattern, because tail behavior differs significantly between, for example, single-node multi-GPU jobs and multi-node distributed jobs. Categories can be determined from requested GPU

count, whether the job uses distributed communication, and whether it performs heavy input preprocessing. For each category and tenant, the controller maintains rolling statistics of observed startup delay, runtime dilation, and completion time. Because training jobs may run for hours, feedback is delayed [62]. To mitigate delayed feedback, the controller also tracks leading indicators such as storage I/O latency, network congestion, and recent scheduling failure rates for similar shapes. These indicators inform the effective capacity estimator C^{eff} and can provide early warning of tail degradation before completion statistics are available.

The controller's data path consists of three main components. The first component is a telemetry collector that aggregates per-resource utilization, fragmentation metrics, and shared-service saturation signals at a cadence on the order of seconds to minutes. Fragmentation metrics for GPUs can be derived from the scheduler's view of free device groups and placement feasibility checks [63]. The second component is a tail estimator that computes empirical quantiles for dilation and startup delay per tenant-category pair, together with uncertainty margins that widen when sample sizes are small or variability increases. The third component is the admission decision engine that computes effective demand inflation γ, slack σ, and then applies the admission inequality using current effective utilization and capacity. The decision is deterministic given inputs, which supports auditability [64].

Deferral semantics matter for user experience. Instead of rejecting, the controller can enqueue jobs in an admission queue per tenant, releasing them when the admission check passes. This makes A_j explicit and controllable. Tenants can be provided with visibility into admission backlogs and expected release times [65]. For interactive workloads, the platform can allow tenants to mark certain jobs as urgency-critical, which increases their utility u_j and makes them less likely to be deferred, subject to global risk budgets. For periodic retraining, the platform can allow jobs to be submitted ahead of deadlines so that controlled admission delay still permits predictable completion.

To evaluate the policy, one can use trace-driven simulation or a prototype on a test cluster. A trace-driven approach is often practical because production tail behavior emerges over long periods and under diverse workloads [66]. The evaluation compares the proposed risk-budgeted admission controller against baselines such as no admission control, simple utilization-threshold gating, and per-tenant quota-based throttling. The primary metrics are p95 and p99 of time-to-start and time-to-finish for each tenant, the fraction of jobs missing deadline thresholds, and cluster utilization. Secondary metrics include fairness across tenants and the stability of admission backlogs.

In a representative trace derived from a marketplace ML platform, arrivals exhibit diurnal patterns and correlated bursts at dataset refresh times [67]. The workload includes a mix of small single-node jobs and large distributed jobs that require multiple nodes. Under no admission control, utilization remains high but p99 time-to-start exhibits large spikes during bursts, and p99 completion time for some tenants exceeds refresh-window thresholds. Simple utilization gating improves mean queueing but often fails to protect tails because it ignores fragmentation and shared-service saturation; p99 delays can remain large even when average GPU utilization is below the threshold, particularly when large jobs cannot be placed due to scattered free GPUs [68]. Quota-based throttling enforces fairness but does not explicitly target tail metrics, and can still permit regimes where correlated bursts cause tail blowups.

Under the proposed policy, effective capacity decreases during periods of high fragmentation and shared-service saturation, which triggers more conservative admission. Virtual backlogs Z_t and G rise when tail violations are observed, increasing inflation and slack and thereby reducing admissions until tail metrics recover. Empirically, this reduces the frequency and duration of tail-degraded regimes [69]. The p99 time-to-start decreases because fewer jobs are admitted into the scheduler queue when placement feasibility is low, reducing contention and failed placement churn. The p99 completion time decreases because admitted jobs experience less interference on network and storage bottlenecks, and because large distributed jobs are less likely to start in suboptimal placements that increase straggler effects. Utilization remains comparable because the controller relaxes conservatism when tails are healthy, admitting more work to fill capacity.

An important observation in such evaluations is that controlled admission delay can increase median latency for some low-priority jobs, because they may be deferred during tail-risk spikes [70]. However, the distribution becomes tighter, and high-percentile latencies improve. In marketplace settings, this trade-off can be acceptable if low-priority jobs are not tied to strict deadlines, and if the platform communicates admission delay explicitly. For high-priority tenants, the controller can be configured with lower sensitivity α and higher utility weights, resulting in fewer deferrals while still benefiting from global tail-risk regulation [71]. This yields differentiated service while maintaining cluster-level predictability.

The policy's dependence on quantile estimates raises concerns about estimation error. Trace-driven results typically show that uncertainty margins are important during regime changes, such as after a software update that affects input pipeline performance or after a seasonal traffic shift that changes training data size. When margins are omitted, the controller may under-react initially, allowing a tail blowup that then causes large virtual backlogs and aggressive throttling [72]. With margins, the controller becomes conservatively reactive earlier, preventing extreme tails at the cost of modest temporary underutilization. Over time, as new samples accumulate, margins shrink and utilization recovers.

Another practical factor is the interaction with the underlying scheduler. If the scheduler already implements gang scheduling for distributed jobs and has strong placement heuristics, fragmentation tails may be reduced, and the admission controller primarily regulates shared-service saturation and burst overload [73]. If the scheduler has weaker placement and experiences frequent retries, the admission controller can provide larger benefits by keeping the queue smaller and reducing placement churn. In both cases, the controller's state inputs should include scheduler-specific signals that correlate with tail behavior, such as placement failure counters, average scheduling cycle time, and the distribution of queued job shapes.

Overall, the evaluation indicates that cluster-level admission control targeted at tail predictability can be effective when it uses conservative effective-load gating and feedback regulation based on observed tail violations [74]. The improvements are not tied to a specific model architecture or training framework; rather, they arise from regulating the shared resource environment in which diverse training jobs execute. In e-commerce marketplace ML platforms, where deadlines and cadence matter, this form of predictability can be as operationally important as raw throughput.

7. Conclusion

Multi-tenant ML training clusters in e-commerce marketplace platforms face a predictability problem that is not well captured by average utilization or mean waiting time. Tail latency emerges from bursty correlated arrivals, multi-resource and topology constraints that induce fragmentation, and interference effects that dilate runtime through network, storage, and synchronization bottlenecks [75]. This paper presented a cluster-level admission control approach that treats tail predictability as a controllable objective by shaping which jobs are admitted into the scheduler under current conditions. The proposed policy combines quantile-aware effective-load estimation with a virtual-queue tail-risk regulator that adapts conservatism based on observed SLO violations, using dynamic demand inflation and slack reservation to avoid sustained operation in tail-degraded regimes.

The formulation decomposes completion latency into admission, queueing, and execution components, which clarifies why admission control can be preferable to uncontrolled queue growth when the goal is predictable high-percentile behavior. The virtual-queue mechanism provides a practical link between percentile-style constraints and enforceable feedback control, and an analytical characterization suggests that stability of the virtual backlogs corresponds to controlling long-run violation rates [76]. Implementation considerations emphasize compatibility with existing schedulers and reliance on telemetry commonly available in ML platforms, including fragmentation indicators and shared-service saturation signals. Trace-driven evaluation discussion highlights the expected trade-offs, where controlled admission delay may increase median latency for some workloads but reduces p99 delays and deadline misses while preserving utilization through adaptive relaxation when tails are healthy.

In marketplace environments where model refresh cadence and workflow timing are operational constraints, cluster-level admission control offers a structured way to trade off utilization and predictability without requiring intrusive changes to training frameworks. The broader implication is that training platforms benefit from service-like control loops that operate on quantile-oriented metrics, complementing placement and scheduling with an outer layer that limits exposure to tail-risk regimes [77].

References

[1] I.-L. Kao, *Effective and Efficient Authentication and Authorization in Distributed Systems*. 5 2019.

[2] E. Wagner, R. Matzutt, J. Pennekamp, L. Bader, I. Bajelidze, K. Wehrle, and M. Henze, "Scalable and privacy-focused company-centric supply chain management," 1 2022.

[3] I. Beschastnikh, P. Liu, A. Xing, P. Wang, Y. Brun, and M. D. Ernst, "Visualizing distributed system executions," *ACM Transactions on Software Engineering and Methodology*, vol. 29, pp. 1–38, 3 2020.

[4] H. Benítez-Pérez, J. L. Ortega-Arjona, P. E. Méndez-Monroy, E. Rubio-Acosta, and O. A. Esquivel-

Flores, *Distributed Systems Modelling*, pp. 41–82. Germany: Springer International Publishing, 8 2018.

[5] E. Johnsen, "Reliable asynchronous communication in distributed systems," 4 2018.

[6] H. Zhang, R. Chen, Z. Tang, K. Cheng, and H. Chen, "Accelerating million-scale in-network lock management using lock fission," *ACM Transactions on Computer Systems*, 11 2025.

[7] A. Arman, P. Bellini, D. Bologna, P. Nesi, G. Pantaleo, and M. Paolucci, "Automating iot data ingestion enabling visual representation.," *Sensors (Basel, Switzerland)*, vol. 21, pp. 8429–8429, 12 2021.

[8] R. Malik, S. Kim, X. Jin, C. Ramachandran, J. Han, I. Gupta, and K. Nahrstedt, "Mlr-index: An index structure for fast and scalable similarity search in high dimensions," in *International Conference on Scientific and Statistical Database Management*, pp. 167–184, Springer, 2009.

[9] S. Kamara, "Encrypted distributed systems," in *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pp. 163–163, ACM, 7 2022.

[10] A. S. Hosseini and S. Staab, "Emotional framing in the spreading of false and true claims," in *Proceedings of the 15th ACM Web Science Conference 2023*, pp. 96–106, ACM, 4 2023.

[11] M. ter Beek, B. Re, M. Viroli, R. Anane, and R. Bahsoon, "Session details: Distributed systems: Ccs track," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ACM, 4 2018.

[12] E. Collini, F. I. Kurniadi, P. Nesi, and G. Pantaleo, "Context-aware retrieval augmented generation using similarity validation to handle context inconsistencies in large language models," *IEEE Access*, pp. 1–1, 1 2025.

[13] V. Kale, *Distributed Systems*, pp. 159–182. CRC Press, 10 2018.

[14] R. K. Ghosh and H. Ghosh, "Global states and termination detection," 2 2023.

[15] S. Soori, B. Can, M. Gurbuzbalaban, and M. M. Dehnavi, "Async: Asynchronous machine learning on distributed systems.," 7 2019.

[16] P. Raith, T. Rausch, A. Furutanpey, and S. Dustdar, "faas-sim: A trace-driven simulation framework for serverless edge computing platforms," *Software: Practice and Experience*, vol. 53, pp. 2327–2361, 10 2023.

[17] N. V. Patel, "Estimating heterogeneous treatment effects of driver incentives in ride-hailing platforms," *Northern Reviews on Algorithmic Research, Theoretical Computation, and Complexity*, vol. 9, no. 11, pp. 16–43, 2024.

[18] A. Fieschi, P. Hirmer, R. Sturm, M. Eisele, and B. Mitschang, "Anonymization use cases for data transfer in the automotive domain," in *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 98–103, IEEE, 3 2023.

[19] K. Davis, M. Schulte, and B. Uekermann, "Enhancing quasi-newton acceleration for fluid-structure interaction," *Mathematical and Computational Applications*, vol. 27, pp. 40–40, 5 2022.

[20] D. Le-Phuoc and M. Hauswirth, *Semantic Stream Processing*, pp. 1505–1513. Springer International Publishing, 2 2019.

[21] A. Charapko, A. Ailijiang, M. Demirbas, and S. S. Kulkarni, "Retroscope: Retrospective monitoring of distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, pp. 2582–2594, 11 2019.

[22] M. Rot and G. Kosec, "Mipro - natural convection of non-newtonian fluids in a differentially heated closed cavity," in *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, pp. 259–264, IEEE, 9 2021.

[23] Q. Liu, D. Du, Y. Xia, P. Zhang, and H. Chen, "The gap between serverless research and real-world systems," in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, pp. 475–485, ACM, 10 2023.

[24] D. Schneider and B. Uekermann, "Efficient partition-of-unity radial-basis-function interpolation for coupled problems," *SIAM Journal on Scientific Computing*, vol. 47, pp. B558–B582, 4 2025.

[25] V. Vijaykumar, R. Chandrasekar, and T. Srinivasan, "An obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs,"

in *2006 IEEE Conference on Cybernetics and Intelligent Systems*, pp. 1–6, IEEE, 2006.

[26] J. Jaskolka, "Evaluating the exploitability of implicit interactions in distributed systems.," 6 2020.

[27] K. Wolsing, A. Saillard, J. Bauer, E. Wagner, C. van Sloun, I. B. Fink, M. Schmidt, K. Wehrle, and M. Henze, "Network attacks against marine radar systems: A taxonomy, simulation environment, and dataset," in *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pp. 114–122, IEEE, 9 2022.

[28] E. Arif, "Implementing a distributed system as a solution to problems in the indosat cooperative," *JURNAL PETIK*, vol. 5, pp. 24–29, 10 2019.

[29] R. Kemp, *Devising – Embodied Creativity in Distributed Systems*, pp. 48–57. Routledge, 9 2018.

[30] G. R. S. Martinez, "Soa distributed systems architecture," 9 2021.

[31] S. B. Kaleel and A. Abhari, "Tafe-ai: A trust-augmented fog and edge ai enabled network integrated with decentralized identity," in *2024 Annual Modeling and Simulation Conference (ANNSIM)*, pp. 1–13, IEEE, 5 2024.

[32] S. Shetty, X. Yuchi, and M. Song, *Moving Target Defense for Distributed Systems*. 4 2018.

[33] T. Srinivasan, R. Chandrasekar, V. Vijaykumar, V. Mahadevan, A. Meyyappan, and A. Manikandan, "Localized tree change multicast protocol for mobile ad hoc networks," in *2006 International Conference on Wireless and Mobile Communications (ICWMC'06)*, pp. 44–44, IEEE, 2006.

[34] A. Ramesh, T. Huang, E. Ruppel, D. Dasari, B. Pourmohseni, F. Smirnov, M. Giani, P. Pazzaglia, C. Shelton, N. Pereira, A. Hamann, D. Ziegenbein, and A. Rowe, "Silverline: Lightweight virtualization and orchestration of distributed systems," in *2025 IEEE 31st Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 375–388, IEEE, 5 2025.

[35] B. Nissen, L. Pschetz, E. Tallyn, A. Polvora, and C. Speed, "Tangible tools for understanding distributed systems," 6 2018.

[36] J. Slak and G. Kosec, "Fast generation of variable density node distributions for mesh-free methods," *WIT transactions on engineering sciences*, vol. 122, pp. 163–173, 9 2018.

[37] L. Zhang and S. Fulton, "Discussion of quantum consensus algorithms," 1 2022.

[38] W. B. Daszczuk, *DepCoS-RELCOMEX - Fairness in Temporal Verification of Distributed Systems*, pp. 135–150. Springer International Publishing, 5 2018.

[39] R. Fronteddu, U. Ardinghi, L. Colombi, S. Dahdal, A. Morelli, M. Tortonesi, C. Stefanelli, and N. Suri, "Semantic information management systems," in *2025 International Conference on Military Communication and Information Systems (ICMCIS)*, pp. 1–9, IEEE, 5 2025.

[40] E. Wagner, F. Basels, J. Bauer, T. Zimmermann, K. Wehrle, and M. Henze, "Caiba: Multicast source authentication for can through reactive bit flipping," in *2025 IEEE 10th European Symposium on Security and Privacy (EuroS&P)*, pp. 710–719, IEEE, 6 2025.

[41] R. Chandrasekar and T. Srinivasan, "An improved probabilistic ant based clustering for distributed databases," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, pp. 2701–2706, 2007.

[42] J. Pennekamp, J. Lohmöller, E. Vlad, J. Loos, N. Rodemann, P. Sapel, I. B. Fink, S. Schmitz, C. Hopmann, M. Jarke, G. Schuh, K. Wehrle, and M. Henze, *Designing Secure and Privacy-Preserving Information Systems for Industry Benchmarking*, pp. 489–505. Germany: Springer Nature Switzerland, 6 2023.

[43] A. A. Hussein, A. AL-Zebari, N. Omar, K. J. Merceedi, A. J. Ahmed, N. O. M. Salim, S. S. Hasan, S. F. Kak, I. M. Ibrahim, H. M. Yasin, and A. A. Salih, "State of art survey for fault tolerance feasibility in distributed systems," *Asian Journal of Research in Computer Science*, pp. 19–34, 9 2021.

[44] I. Schagaev, *Distributed Systems: Maximizing Resilience*, pp. 325–342. Springer International Publishing, 7 2024.

[45] J. L. Clark, "A design approach for a distributed system to handle chinese," *Proceedings of the Annual*

Conference of CAIS / Actes du congrès annuel de l'ACSI, 3 2022.

[46] L. Januário, B. H. Pereira, B. J. Mello, D. R. Melo, F. Viel, and C. A. Zeferino, "Fpga-based platform for food classification using hyperspectral images," in *2025 32nd IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4, IEEE, 11 2025.

[47] *Meta Heuristic Algorithms for Advanced Distributed Systems*. Wiley, 3 2024.

[48] I. Schagaev, *Distributed Systems: Maximizing Resilience*, pp. 249–266. Springer International Publishing, 7 2019.

[49] N. V. Patel, "Estimating the causal effects of pricing interventions in two-sided marketplaces under interference and spillovers," *Frontiers in Health Informatics*, vol. 13, no. 8, pp. 7572–7601, 2024.

[50] D. G. Balreira, T. da Silva Araújo, and F. Petrillo, "Visualizing kubernetes distributed systems: An exploratory study," in *2023 IEEE Working Conference on Software Visualization (VISSOFT)*, pp. 12–22, IEEE, 10 2023.

[51] Y. Mnaoui, A. Najoua, and H. Ouajji, *Automatic English Proficiency Assessment for Air Traffic Controllers in Emergencies*, pp. 996–1002. Springer International Publishing, 2 2022.

[52] W. B. Daszczuk, *Fairness in Distributed Systems Verification*, pp. 139–159. Germany: Springer International Publishing, 3 2019.

[53] Y. Teng, L. Liu, J. Qi, H. Pan, and C. Fan, "Vmr-tree: Efficient and verifiable location-based knn queries on blockchain," in *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 342–351, IEEE, 11 2023.

[54] F. Solowjow, A. Mehrjou, B. Schölkopf, and S. Trimpe, "Minimum information exchange in distributed systems.," 5 2018.

[55] S. Zhu, "Chat application with distributed system," 5 2020.

[56] *On Convex Embedding and Control Design for Nonlinear Homogeneous Systems*, 6 2021.

[57] S. Ahriz, N. Benmoussa, A. E. Yamami, K. Mansouri, and M. Qbadou, "An elaboration of a strategic alignment model of university information systems based on sam model," *Engineering, Technology & Applied Science Research*, vol. 8, pp. 2471–2476, 2 2018.

[58] R. Chandrasekar, R. Suresh, and S. Ponnambalam, "Evaluating an obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 International Conference on Advanced Computing and Communications*, pp. 628–629, IEEE, 2006.

[59] "Trusted working copies for distributed systems engineering," 1 2022.

[60] P. Dubey, D. Srivastava, K. Singh, and M. V. Singh, "Middleware architecture for microservices based distributed system," in *2023 13th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 200–207, IEEE, 1 2023.

[61] Z. Wang, M. Goudarzi, J. Aryal, and R. Buyya, *Container Orchestration in Edge and Fog Computing Environments for Real-Time IoT Applications*, pp. 1–21. Springer Nature Singapore, 9 2022.

[62] L. Bull and H. Liu, "A generalised dropout mechanism for distributed systems.," *Artificial life*, vol. 29, pp. 146–152, 5 2023.

[63] G. A. M. Sborz, G. A. Pohl, F. Viel, and C. A. Zeferino, "Sbcci - a custom processor for an fpga-based platform for automatic license plate recognition," in *Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design*, pp. 15–6, ACM, 8 2019.

[64] D. Telezhenko and O. Tolstoluzka, "A conceptual model for synthesizing the architecture of virtual distributed systems," *Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems»*, pp. 58–65, 10 2022.

[65] A. Rashidov, A. Akhatov, I. Aminov, D. Mardonov, and A. Dagur, *Distribution of data flows in distributed systems using hierarchical clustering*, pp. 207–212. CRC Press, 6 2024.

[66] N. V. Patel, "Selection bias in two-sided e-commerce marketplaces: A framework for propensity score

matching implementation," *Journal of Business Intelligence Systems and Computational Social Science Applications*, vol. 11, no. 5, pp. 1–20, 2021.

[67] A. Mampage and R. Buyya, "Cloudsimsc: A toolkit for modeling and simulation of serverless computing environments," in *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pp. 550–557, IEEE, 12 2023.

[68] F. Kargl, B. Erb, and C. Bösch, *Defining Privacy*, pp. 461–463. Springer International Publishing, 7 2022.

[69] W. Liao, L. Han, and P. Chen, "Design and optimization of asymmetric encryption scheme based on blockchain and its application in privacy protection of internet of vehicles," in *2024 IEEE International Conference on High Performance Computing and Communications (HPCC)*, pp. 1090–1097, IEEE, 12 2024.

[70] G. Heinrich and I. Frosio, "Metaoptimization on a distributed system for deep reinforcement learning," 1 2019.

[71] R. K. Ghosh and H. Ghosh, *Distributed Systems*. Wiley, 2 2023.

[72] M. L. Nunes, "Backscatter readers for single and distributed systems," 12 2019.

[73] Z. Kazhmaganbetova, S. Imangaliyev, and A. Sharipbay, "Machine learning for the communication optimization in distributed systems," *International Journal of Engineering & Technology*, vol. 7, pp. 47–, 9 2018.

[74] R. Malik, C. Ramachandran, I. Gupta, and K. Nahrstedt, "Samera: a scalable and memory-efficient feature extraction algorithm for short 3d video segments.," in *IMMERSCOM*, p. 18, 2009.

[75] A. Saleh, "Privacy-protection in cooperative distributed systems," 8 2018.

[76] X. Wei, Z. Huang, T. Sun, Y. Hao, R. Chen, M. Han, J. Gu, and H. Chen, "Phoenixos: Concurrent os-level gpu checkpoint and restore with validated speculation," in *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pp. 996–1013, ACM, 10 2025.

[77] K. Smirnov, B. Faustov, and I. Faustova, "Distributed system for calculating the velocity field," *Izvestiya vysshikh uchebnykh zavedenii. Fizika*, vol. 64, pp. 100–105, 2 2020.