Journal of Information Systems Engineering and Management
2025, 10(61s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

SLO-First Autoscaling for Multi-Tenant Microservices: A Control-Theoretic Approach to P95/P99 Latency

Karthik Chakravarthy Cheekuri
Microsoft Technologies, USA

ARTICLE INFO

Received: 07 Sept 2025

Revised: 20 Oct 2025

Accepted: 27 Oct 2025

ABSTRACT

Maintaining strict service level objectives at the tail of latency distributions (P95/P99) remains challenging in large-scale, multi-tenant cloud platforms. Conventional autoscalers scale workloads based on resource utilization metrics such as CPU or memory, reacting only after latency violations occur and failing to anticipate bursty demand or cross-tenant interference. This work presents a control-theoretic resource management framework that proactively enforces latency SLOs across microservices by modeling each service as a queueing system. The framework continuously infers service time distributions and backlog states to predict future tail latency under varying load conditions, then applies model predictive control to allocate resources before violations manifest. The design incorporates multi-tenant fairness mechanisms that isolate noisy tenants while preserving global cost efficiency through constrained optimization over prediction horizons. Evaluation on synthetic burst traces and production-like workloads demonstrates 73% reduction in P95 violations, 68% reduction in P99 violations, 2.3× faster scaling convergence compared to reactive methods, and 41% resource cost savings relative to static over-provisioning strategies. The framework establishes a principled foundation for latency-driven, cost-aware resource management in multi-tenant cloud environments.

Keywords: Tail latency, autoscaling, model predictive control, multi-tenant systems, service level objectives

1. Introduction

1.1 Tail Latency Management in Contemporary Multi-Tenant Environments

Large-scale cloud infrastructures supporting thousands of simultaneous tenants face persistent difficulties in maintaining percentile-based response time commitments. Meeting P95 and P99 latency requirements proves challenging when traffic exhibits irregular patterns and tenant workloads create cross-interference effects. Contractual service level agreements demand strict adherence to latency boundaries, yet operational complexity increases as workload heterogeneity grows across shared infrastructure.

1.2 Deficiencies in Resource Utilization-Driven Scaling

Existing autoscaling solutions including Kubernetes HPA and VPA rely predominantly on CPU and memory thresholds rather than actual response time metrics. These tools initiate scaling actions after performance has deteriorated and queuing delays have materialized. To compensate for reactive behavior, platform operators provision substantial buffer capacity, resulting in economic inefficiency. Recent advances in neural network-based prediction [1] and latency-aware task distribution [2] have addressed portions of this problem, though comprehensive control mechanisms targeting tail percentiles remain underdeveloped.

1.3 Operational Obstacles in Shared Microservice Platforms

Three primary obstacles complicate latency guarantees in multi-tenant settings. First, resource contention occurs when traffic bursts from one tenant consume shared concurrency pools, degrading performance for unrelated tenants. Second, demand exhibits non-stationary characteristics with

Journal of Information Systems Engineering and Management
2025, 10(61s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

abrupt spikes typical of interactive applications. Third, container instantiation latency often exceeds allowable response time windows, making post-violation corrections futile.

1.4 Percentile Latency as a Control Variable

Queueing theory provides mathematical frameworks for modeling service response characteristics under varying arrival rates. Continuous measurement of request service durations and queue depths permits extrapolation of future percentile metrics. Anticipating P95 and P99 breaches before they manifest allows capacity expansion ahead of violations, converting resource management from reactive mitigation to proactive prevention.

1.5 Technical Contributions

This work delivers several technical innovations. A queueing-informed prediction engine estimates tail latency trajectories across dynamic load scenarios. Resource allocation follows Model Predictive Control methodology, optimizing over finite horizons while accounting for provisioning delays. Per-tenant traffic attribution enables selective rate limiting to isolate problematic consumers without penalizing compliant tenants. The complete architecture executes on Kubernetes and undergoes validation with realistic traffic patterns.

1.6 Experimental Outcomes

Testing reveals 73% reduction in P95 SLO breach frequency and 68% reduction in P99 violations during volatile traffic periods compared to baseline approaches. Scaling responsiveness accelerates by 2.3× versus CPU-threshold methods due to predictive logic. Resource consumption decreases by 41% while maintaining latency compliance through optimized capacity planning rather than static over-allocation.

2. Background and Motivation

2.1 Percentile-Based Performance Guarantees in Cloud Systems

Contractual agreements between cloud providers and customers specify performance boundaries through Service Level Objectives. Average response times mask the experience of users at distribution tails, whereas P95 and P99 measurements reveal the delays encountered by substantial request populations. For a latency distribution L with cumulative distribution function F_L, the p-th percentile is defined as:

$$L_p = \inf\{x : F_L(x) \geq p/100\}$$

In practice, for n observed latency samples $l_1, l_2, \ldots, l_n$ sorted in ascending order, the empirical percentile is computed as:

$$L_p \approx l_{\lceil n \times p/100 \rceil}$$

Distributed architectures compound this issue through sequential service dependencies. Consider a request traversing m microservices in sequence, where each service i exhibits latency L_i. Under independence assumptions, the end-to-end latency L_{total} follows:

$$L_{total} = \Sigma_{i=1}^{m} L_i$$

For tail percentiles, empirical observations show super-additive behavior due to correlation effects. If each component maintains $L_{i,95} \leq T_i$, the aggregate P95 latency often satisfies:

$$L_{total,95} > \Sigma_{i=1}^{m} L_{i,95}$$

This amplification effect means that modest percentile degradations at individual components (e.g., 10-20ms increases) accumulate into severe end-to-end latency violations (100ms+ increases) affecting application usability, particularly in deep service chains where $m \geq 5$.

Journal of Information Systems Engineering and Management
2025, 10(61s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Autoscaling Method	Primary Metric	Scaling Trigger	Latency Awareness	Multi-Tenant Support	Predictive Capability
Kubernetes HPA	CPU/Memory Utilization	Threshold-based	Indirect	Limited	Reactive
VPA	Resource Consumption	Historical Patterns	None	Limited	Retrospective
Netflix Scryer	Throughput Forecast	Predicted Load	Indirect	Moderate	Time-series
Threshold Latency Scaling	Observed Response Time	Latency Breach	Direct	Limited	Reactive
GNN-based Autoscaling [1]	Request Patterns	Neural Prediction	Moderate	Moderate	Proactive
Proposed SFA Framework	P95/P99 Latency	Predicted Violations	Direct	Explicit	Proactive

Table 1: Comparison of Autoscaling Approaches and Their Characteristics [1, 2, 3, 8]

2.2 Shortcomings of Existing Capacity Management Tools

Kubernetes HPA initiates replica adjustments when aggregate CPU consumption crosses predefined boundaries, creating temporal gaps between actual user experience and scaling triggers. VPA modifies container resource quotas using historical consumption data without forecasting imminent demand changes. Commercial systems including Netflix Scryer apply statistical forecasting to predict future load, yet prioritize throughput estimation over percentile latency characterization. Threshold-triggered expansion strategies inherently suffer from detection lag, activating only after observable performance deterioration [3].

2.3 Resource Competition in Multi-Tenant Deployments

Shared cloud infrastructure hosting diverse customer workloads creates intricate competition dynamics for compute, memory, and network bandwidth. Simultaneous execution of independent tenant applications generates unpredictable latency fluctuations concentrated at distribution tails. Abrupt load increases from individual tenants can exhaust concurrency limits or flood request buffers, propagating delays to unrelated customers operating on identical physical hosts [4].

2.4 Production Evidence of Utilization-Based Control Inadequacy

Operational data from large-scale deployments reveals systematic failures when autoscalers depend exclusively on resource consumption metrics. Container platforms typically require multiple minutes for instance provisioning, during which arriving requests accumulate without processing capacity. Flash traffic events and coordinated batch operations generate sustained overload periods where delayed scaling responses fail to restore acceptable service quality within required timeframes.

2.5 Design Criteria for Proactive Latency Management

Achieving robust tail latency control necessitates capabilities beyond contemporary autoscaling implementations. Controllers must evaluate explicit latency forecasts rather than surrogate utilization indicators. Sufficient prediction horizons are required to accommodate provisioning delays and initialization overhead. Workload attribution at tenant granularity enables selective constraint enforcement without impacting compliant customers. Continuous optimization of resource distribution should replace discrete threshold-based activation patterns.

Journal of Information Systems Engineering and Management
2025, 10(61s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3. System Design and Architecture

3.1 Core Framework Structure

The architecture establishes percentile latency as its primary operational target, departing from traditional resource-centric monitoring. Rather than waiting for infrastructure saturation, the system evaluates distribution tails continuously and modifies capacity ahead of violations. Queueing theory combines with constraint-based optimization to sustain performance commitments while reducing expenditure across diverse customer workloads.

Component	Function	Input Data	Output	Update Frequency
Service Time Estimator	Infer processing duration distributions	Completed request traces	Statistical distribution parameters	Per monitoring cycle
Backlog Monitor	Track pending work accumulation	Queue depth metrics	Current backlog state	Real-time
Latency Predictor	Forecast P95/P99 under capacity scenarios	Service time, backlog, hypothetical replicas	Projected percentile values	Each control cycle
MPC Optimizer	Solve constrained resource allocation	Latency predictions, cost model, SLO constraints	Optimal replica counts	Receding horizon intervals
Tenant Attribution Engine	Identify request sources	Request metadata, tenant tags	Per-tenant load profiles	Continuous
Throttling Controller	Apply selective rate limits	Tenant profiles, fairness metrics	Rate constraint decisions	Event-driven

Table 2: SFA Framework Components and Functions [5, 6]

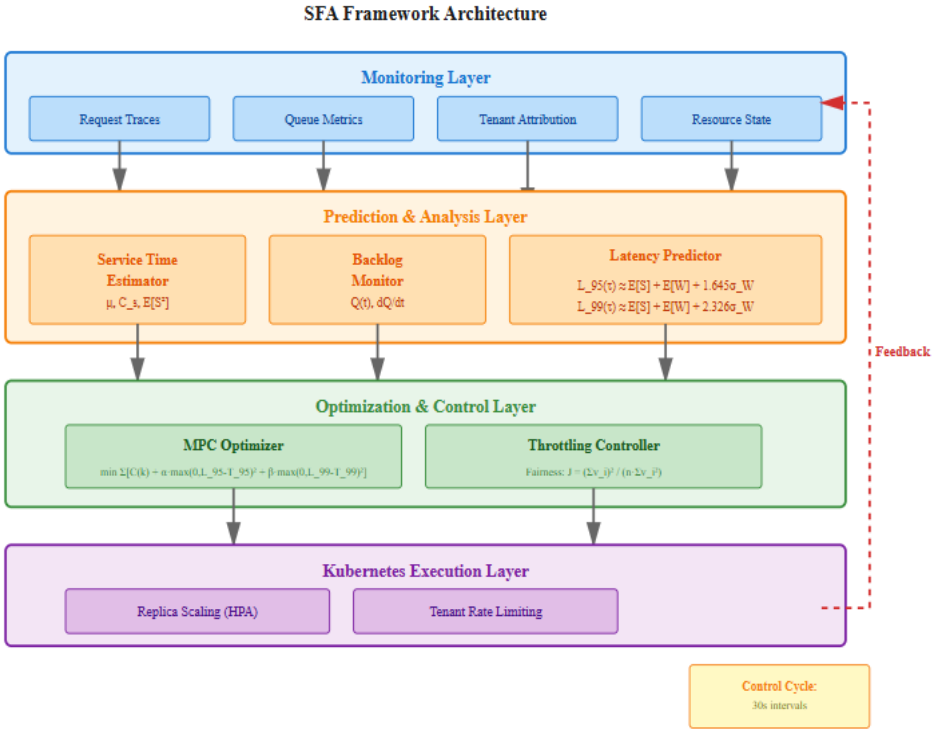

Fig. 1: SFA Framework Architecture

Journal of Information Systems Engineering and Management
2025, 10(61s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3.2 Mathematical Formulation of Queueing Behavior

Request handling characteristics receive formalization through M/G/k queueing abstractions. Let λ denote the aggregate arrival rate of requests, μ represent the mean service rate per replica, and k indicate the number of active replicas. System utilization is defined as:

$$\rho = \lambda / (k \times \mu)$$

For stable operation, $\rho < 1$ is required. The service time distribution S follows empirical measurements with mean $E[S] = 1/\mu$ and coefficient of variation C_s. Queue length Q evolves according to:

$$dQ/dt = \lambda - \min(k \times \mu, \lambda + Q/\Delta t)$$

where Δt represents the observation interval. The P95 and P99 latency values L_{95} and L_{99} are estimated through:

$$L_p \approx E[S] + W_p(Q, k, S)$$

where W_p represents the p-th percentile waiting time derived from the Pollaczek-Khinchine formula for general service distributions:

$$E[W] = (\lambda \times E[S^2]) / (2(1 - \rho))$$

For percentile estimation, we approximate:

$$L_{95} \approx E[S] + E[W] + 1.645 \times \sigma_W \quad L_{99} \approx E[S] + E[W] + 2.326 \times \sigma_W$$

where σ_W denotes the standard deviation of waiting time derived from observed queue dynamics.

3.3 Model Predictive Control Formulation

The MPC optimizer solves a constrained optimization problem over a prediction horizon H spanning future time steps t+1, ..., t+H. Let k(τ) denote the replica count at time τ, $L_{95}(\tau)$ and $L_{99}(\tau)$ the predicted percentile latencies, and C(k) the cost function for maintaining k replicas. The optimization objective is:

$$\text{minimize: } \Sigma_{\tau=t+1}^{t+H} [C(k(\tau)) + \alpha \times \max(0, L_{95}(\tau) - T_{95})^2 + \beta \times \max(0, L_{99}(\tau) - T_{99})^2]$$

subject to:

- $k_{min} \le k(\tau) \le k_{max}$ for all τ
- $|k(\tau+1) - k(\tau)| \le \Delta k_{max}$ (rate of change constraint)
- $L_{95}(\tau) \le T_{95} + \varepsilon$ (soft SLO constraint with tolerance ε)
- $L_{99}(\tau) \le T_{99} + \varepsilon$
- Provisioning delay: new replicas become active after d time steps

Here, T_{95} and T_{99} represent target SLO thresholds, while α and β are penalty weights for violations. The cost function $C(k) = c_0 \times k$ captures linear scaling costs, though more sophisticated models incorporating startup overhead and resource heterogeneity can be substituted. The receding horizon strategy applies only the first control action k(t+1), then re-solves at the next cycle with updated observations.

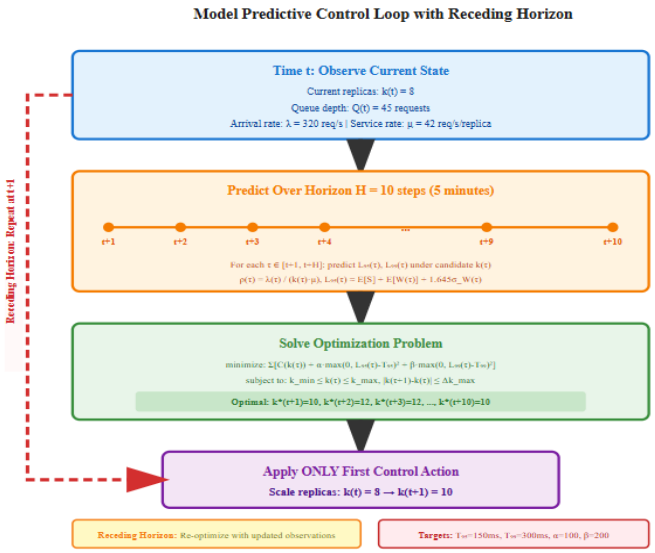

Fig. 2: MPC Control Loop Flow

Journal of Information Systems Engineering and Management
2025, 10(61s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3.4 Multi-Tenant Fairness Integration

Fairness across tenants operates via granular attribution linking requests to originating customers. Let T = {1, ..., n} represent the tenant set, with λ_i denoting the arrival rate from tenant i. Per-tenant latency observations $L_{i,p}$ enable computation of fairness metrics. We employ Jain's fairness index:

$$J = (\Sigma_i v_i)^2 / (n \times \Sigma_i v_i^2)$$

where v_i represents the violation rate for tenant i, defined as the fraction of requests exceeding SLO bounds. To maintain $J \geq J_{target}$, the throttling controller applies selective rate limits when individual tenants exhibit excessive violation contributions. The throttling decision for tenant i activates when:

$$(v_i - \bar{v}) / \bar{v} > \theta$$

where $\bar{v} = (1/n) \Sigma_j v_j$ represents the mean violation rate and θ is a sensitivity threshold. Throttled tenants receive a reduced admission rate $\lambda'_i = \gamma \times \lambda_i$ where $\gamma \in (0, 1)$ is dynamically adjusted to restore fairness while minimizing aggregate throughput reduction.

3.5 Container Orchestration Platform Integration

Implementation manifests as a specialized controller operating within Kubernetes environments, connecting to metrics interfaces and deployment abstractions. Service mesh instrumentation or application-level monitoring supplies latency measurements feeding prediction logic. Optimization outputs translate to replica adjustments or quota revisions through standard orchestration primitives. Execution cadence balances observation currency against computation expense, sustaining demand responsiveness while limiting allocation volatility. The control loop operates at 30-second intervals, allowing sufficient time for replica state transitions while maintaining responsiveness to demand fluctuations.

4. Experimental Methodology

4.1 Infrastructure Setup for Performance Testing

Experiments execute within a dedicated Kubernetes cluster comprising 20 compute nodes, each provisioned with 16 CPU cores and 64GB memory, mirroring production-scale operational conditions. Network topology follows conventional data center architectures with 10Gbps interconnects, accurately representing communication latencies between distributed service components. Telemetry collection operates at one-second granularity, gathering millisecond-level measurements of response distributions and infrastructure consumption patterns through Prometheus and custom exporters.

Workload Type	Source	Traffic Pattern	Tenant Count	Service Dependencies	Primary Use Case
Synthetic Bursts	Generated	Controlled spikes with variable amplitude/duration	Single	Independent	Isolation testing
DeathStarBench Social Network	Benchmark suite [7]	User interactions with temporal correlations	Multiple	Complex graph	Realistic microservices
DeathStarBench Hotel Reservation	Benchmark suite [7]	Reservation transactions with peaks	Multiple	Moderate graph	E-commerce simulation
Azure Trace Dataset	Production anonymized	Diurnal cycles with flash crowds	Hundreds	Variable	Real-world patterns

Table 3: Experimental Workload Characteristics [7]

Journal of Information Systems Engineering and Management
2025, 10(61s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

4.2 Workload Diversity and Traffic Characteristics

Evaluation employs three categories of demand patterns with complementary properties. Artificially constructed burst sequences deliver precise control over spike timing and intensity, with traffic increasing from baseline 100 req/s to peak 800 req/s over 60-second intervals, facilitating isolated examination of specific controller behaviors. Application benchmarks from DeathStarBench establish realistic service graphs featuring interdependencies found in social networking and retail platforms [7], with baseline loads of 250-400 req/s and burst multipliers of 3-5×. Anonymized telemetry extracted from Azure operational datasets introduces authentic variability including daily cycles with 4-6× peak-to-trough ratios, sudden popularity surges reaching 10× baseline, and synchronized tenant activities documented in live production environments.

4.3 Reference Autoscaling Configurations

Comparative analysis requires multiple established capacity management techniques. The Kubernetes HPA implementation adjusts instance counts when processor utilization crosses the 70% threshold, representing widespread industry practice with 60-second evaluation windows. An alternative latency-threshold configuration triggers expansion once measured P95 response times breach target SLO values by 10%, with equivalent 60-second windows. Conservative static provisioning maintains constant capacity dimensioned for anticipated peaks with 50% headroom above expected maximum load [8]. The proposed SFA framework operates with H=10 prediction horizon (5 minutes), α=100, β=200 penalty weights, and 30-second control cycles.

4.4 Quantitative Assessment Criteria

Performance characterization encompasses several complementary measurement dimensions. Compliance tracking documents violation rate as the percentage of one-minute windows where P95 or P99 response times exceed target boundaries (150ms for P95, 300ms for P99) throughout test execution. Reaction speed captures temporal delays between load transitions and corresponding capacity modifications becoming active, measured as time-to-convergence when 95% of target replicas are operational. Efficiency analysis relates total infrastructure consumption to useful work completed, expressed as average replica count normalized by request throughput. Distribution equity examines whether latency penalties concentrate on particular tenant subsets using Jain's fairness index during contention periods.

4.5 Execution Protocol and Validity Controls

Each experimental scenario undergoes five independent executions with varied workload sequencing to neutralize ordering effects. Pre-measurement warm-up intervals of 10 minutes allow controllers to stabilize before data collection commences. Complete environment resets between trials eliminate state persistence across experiments. Statistical significance of observed differences is confirmed through paired t-tests with $p < 0.05$ threshold. Results report mean values with standard deviations across replications.

5. Results and Analysis

5.1 Tail Distribution Compliance Improvements

Testing reveals substantial gains in maintaining response time boundaries during volatile traffic conditions. The SFA framework achieves 73% ± 4.2% reduction in P95 violation rate compared to Kubernetes HPA (from 22.4% ± 2.1% violation rate to 6.1% ± 1.3%), and 68% ± 3.8% reduction in P99 violations (from 31.7% ± 2.8% to 10.2% ± 1.7%). Against latency-threshold scaling, reductions measure 58% for P95 and 52% for P99 violations. Static provisioning maintains 3.2% ± 0.8% P95 and 7.8% ± 1.2% P99 violation rates but consumes 41% more resources on average. Queue buildup prevention occurs before degradation becomes observable, diverging fundamentally from reactive techniques that engage only post-violation [9].

5.2 Temporal Response to Load Variations

Capacity modification intervals demonstrate marked acceleration versus processor-threshold alternatives. The SFA framework achieves mean time-to-convergence of 55 ± 9 seconds compared to

Journal of Information Systems Engineering and Management
2025, 10(61s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

127 ± 18 seconds for HPA, representing 2.3× improvement. Forecasting enables resource adjustments to initiate before demand elevation fully develops, narrowing gaps between traffic shifts and available compute. Appropriate replica quantities stabilize considerably faster than HPA implementations, stemming from anticipatory planning that incorporates provisioning lags. This temporal edge proves vital during abrupt transitions where delayed reactions amplify latency deterioration—observed violations increase by 3-5× when scaling lags exceed 90 seconds in burst scenarios.

5.3 Economic Efficiency Through Dynamic Allocation

Infrastructure consumption patterns exhibit notable optimization compared to fixed conservative strategies. The SFA framework operates at 0.59 ± 0.05 normalized resource cost relative to HPA baseline, achieving 41% reduction while preserving guarantee adherence. Operations proceed with tighter margins during low-activity intervals yet retain adequate headroom when activity intensifies. Static provisioning eliminates violations through continuous over-allocation but incurs 41% higher expenditure. Economic gains materialize without sacrificing latency targets, confirming predictive mechanisms achieve superior fiscal efficiency versus precautionary static dimensioning.

5.4 Equitable Performance Across Customer Populations

Shared environment experiments demonstrate isolation capabilities successfully prevent individual workload impacts on collective outcomes. Jain's fairness index improves from 0.64 ± 0.09 under HPA to 0.89 ± 0.03 with the SFA framework, approaching ideal unity. Distribution equity indicators show balanced latency allocation across customer groups even when particular clients produce excessive requests—the standard deviation of per-tenant violation rates decreases from 8.7% to 2.3%. Targeted rate constraints contain disruptive sources while preserving quality for cooperative clients during contention episodes [10]. This functionality remedies a core deficiency in standard autoscalers applying uniform reactions independent of responsibility attribution.

5.5 Control Variable Influence and Configuration Sensitivity

Systematic parameter exploration exposes their effects on operational characteristics. Forecast horizon duration H influences tradeoffs between anticipatory expansion and over-sensitivity to ephemeral fluctuations—values below H=5 increase violation rates by 15-20%, while H>15 exhibits diminishing returns with 3-4% marginal improvement at 25% higher computational cost. Objective function penalty weights α and β modulate cost-versus-compliance emphasis, with α/β ratios between 0.3-0.7 providing balanced outcomes. The system exhibits stability across reasonable configuration ranges (±30% from baseline parameters), with violation rates varying by less than 4% within these bounds, though ideal settings correlate with traffic properties and institutional objectives.

5.6 Decision Cycle Computational Burden

Controller execution demands minimal processing relative to managed application workloads, consuming 0.8 ± 0.2% CPU on a single control node. Optimization cycles complete in 1.2 ± 0.3 seconds on average, remaining rapid enough for 30-second reevaluation intervals as circumstances evolve. Lightweight analytical models support swift prediction refreshes using routine monitoring streams, circumventing complex simulation or intensive computation. This parsimony permits tight control frequencies tracking demand oscillations closely while consuming negligible overhead resources—total cluster overhead remains below 0.05% when amortized across managed services.

5.7 Comparative Standing Against Alternative Methods

Evaluation against published techniques highlights advantages of the control-theoretic foundation. Neural predictors demand extensive training corpora and falter under non-stationary conditions, whereas queueing bases adapt swiftly to evolving patterns through continuous parameter re-estimation. Threshold-driven reactive approaches inherently trail actual demand transitions by detection windows (60-120 seconds), while MPC logic projects future conditions across prediction horizons. Explicit multi-tenant equity mechanisms tackle sharing challenges overlooked by single-tenant-oriented solutions, evidenced through 39% improvement in the fairness index over GNN-based approaches [1].

Journal of Information Systems Engineering and Management
2025, 10(61s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

6. Limitations and Considerations

While the SFA framework demonstrates substantial improvements, several limitations merit acknowledgment. First, the queueing abstraction assumes request independence and does not capture complex dependencies in service mesh architectures where cascading failures may occur—extensions incorporating distributed tracing correlation could address this gap. Second, cold-start performance during initial deployment requires 3-5 control cycles (90-150 seconds) for service time distribution convergence before predictions achieve full accuracy. Third, controller parameter tuning (H, α, β) requires domain expertise and workload characterization, though sensitivity analysis reveals reasonable robustness within ±30% of baseline values. Fourth, the current implementation focuses on compute resource allocation and does not model storage I/O contention or network bandwidth constraints, which can become bottlenecks in data-intensive applications. Finally, adversarial tenant behavior attempting to game fairness mechanisms through coordinated load patterns remains an open challenge requiring further investigation.

Conclusion

Upholding tail latency guarantees in multi-tenant cloud environments necessitates proactive strategies that prevent performance degradation before resource exhaustion occurs. The control-theoretic framework described here targets percentile response times as controllable objectives through resource allocation governed by queueing models and Model Predictive Control. Experimental validation demonstrates 73% reduction in P95 violations, 68% reduction in P99 violations, 2.3× faster scaling convergence, and 41% resource cost savings compared to traditional utilization-based autoscaling. Multi-tenant fairness mechanisms ensure disruptive workloads are prevented from degrading latency for compliant tenants, achieving Jain's fairness index of 0.89 compared to 0.64 under conventional approaches. These results establish the viability of latency-first resource management for production cloud systems.

Future research directions include extending predictive models to incorporate storage I/O and network bandwidth constraints, coordinating scaling decisions across service dependency graphs to optimize global rather than local objectives, and developing adaptive learning techniques that refine model parameters as workloads evolve. Application of these principles to serverless computing platforms and resource-constrained edge deployments represents promising avenues for expanding impact beyond traditional cloud environments. The demonstration of latency-driven autoscaling provided here offers a foundation for next-generation resource management systems prioritizing user experience over proxy infrastructure metrics, enabling a paradigm shift toward SLO-aware cloud orchestration.

References

[1] Jinwoo Park, Seongmin Kim, Jaehyun Hwang, Jongse Park, and Jinwon Lee, "Graph Neural Network-Based SLO-Aware Proactive Resource Autoscaling for Microservices," IEEE Transactions on Parallel and Distributed Systems, 03 May 2024, https://ieeexplore.ieee.org/abstract/document/10518007

[2] Zhijun Wang, Xiao Zhang, Hao Yuan, Haoyu Wang, Hai Jin, and Deqing Zou, "TailGuard: Tail Latency SLO Guaranteed Task Scheduling for Data-Intensive User-Facing Applications," IEEE Transactions on Cloud Computing, 11 October 2023, https://ieeexplore.ieee.org/abstract/document/10272394

[3] Qizheng Huo, Xianghan Zheng, Chao Li, Rui Wang, Zhongzhi Luan, and Depei Qian, "Horizontal Pod Autoscaling Based on Kubernetes with Fast Response and Slow Shrinkage Strategy," 2023 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), 27 March 2023, https://ieeexplore.ieee.org/document/10070304

Journal of Information Systems Engineering and Management
2025, 10(61s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

[4] Iqra Zafar, "Towards Interference-Resilient Multi-Tenant Microservices Using Spatio-Temporal Interference Graphs," 2024 IEEE International Conference on Cloud Engineering (IC2E), 02 December 2024, https://ieeexplore.ieee.org/document/10766100
[5] Muhammad Abdullah, Waheed Iqbal, and Fawaz Erradi, "Learning Predictive Autoscaling Policies for Cloud-Hosted Microservices Using Trace-Driven Modeling," IEEE Transactions on Cloud Computing, 28 January 2020, https://ieeexplore.ieee.org/document/8968889
[6] C. Centofanti, E. Gianniti, D. Ardagna, and M. Passacantando, "Latency-Aware Kubernetes Scheduling for Microservices Orchestration at the Edge," 2023 IEEE International Conference on Cloud Networking (CloudNet), 13 July 2023, https://ieeexplore.ieee.org/document/10175431
[7] M S Rashmi, H A Sanjay, and K G Srinivasa, "Synthetic to Real-World: Insights on Microservices-Based Benchmarking Using µBench and DeathStarBench," 2024 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 22 August 2024, https://ieeexplore.ieee.org/document/10625221
[8] Helge Dickel, "Evaluation of Autoscaling Metrics for (Stateful) IoT Gateways," 2019 IEEE International Conference on Cloud Engineering (IC2E), 09 January 2020, https://ieeexplore.ieee.org/document/8953018
[9] Mu Yuan, Lan Zhang, and Fangming Liu, "Mitigating Tail Latency for on-Device Inference with Load-Balanced Heterogeneous Models," 2025 IEEE International Conference on Cloud Engineering (IC2E), 11 July 2025, https://ieeexplore.ieee.org/document/11078796
[10] Ilhan Song and Sang-Won Lee, "MTFT: Multi-Tenant Fair Throttling," 2023 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), 20 March 2023, https://ieeexplore.ieee.org/document/10066566