Efficient Resource Allocation in Multi-Tenant Edge Computing Clusters

by

Daniel K. Harrington

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2026

To my parents, who taught me to ask why.

Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

This work was carried out under the supervision of

Dr. Amara Osei, Professor of Computer Science,
University of Waterloo.

Abstract

The rapid proliferation of edge computing has introduced significant challenges in resource allocation across geographically distributed, multi-tenant clusters. This thesis presents **EdgeSched**, a novel scheduling framework that combines predictive workload modeling with latency-aware bin packing to achieve efficient resource utilization across heterogeneous edge nodes.

We formulate the multi-tenant edge allocation problem as a constrained optimization in which latency service-level agreements (SLAs), node capacity, and workload colocation interference are jointly modeled. Drawing on techniques from stochastic optimization and online algorithms, we prove that EdgeSched achieves a competitive ratio of $O(\log n)$ under adversarial workloads, while maintaining sub-millisecond scheduling latency on commodity hardware.

The framework is evaluated on a 64-node edge cluster running representative workloads drawn from video analytics, IoT stream processing, and inference serving. EdgeSched reduces tail latency by up to 42% and improves aggregate node utilization by 31% compared to state-of-the-art schedulers, all while respecting per-tenant isolation guarantees. We further demonstrate the system's robustness under node failures and dynamic workload reshuffling.

These results establish EdgeSched as a practical foundation for resource management in next-generation edge infrastructure, with implications for 5G services, autonomous systems, and large-scale IoT deployments.

Keywords: edge computing, resource allocation, scheduling, multi-tenancy, optimization

Acknowledgements

I am deeply grateful to my supervisor, Dr. Amara Osei, whose patience, insight, and unwavering belief in this work made it possible. Her ability to see the forest when I was lost among the trees is a gift I will carry with me.

I thank the members of my examining committee — Dr. Priya Malhotra, Dr. James Whitfield, Dr. Lena Kovacs, and Dr. Toshiro Yamamoto — for their thoughtful questions and generous engagement with this research.

My colleagues in the Distributed Systems Lab at Waterloo transformed the long hours into something approaching joy. Special thanks to Sarah Chen for the late-night debugging sessions that sometimes produced code and always produced friendship.

Funding for this research was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Ontario Graduate Scholarship program, and a research grant from Nimbus Computing. The compute infrastructure was generously provided by Vertex Cloud Services.

Finally, my family. Thank you for everything.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation and Problem Statement

The shift toward edge computing represents one of the most significant architectural transitions in distributed systems since the advent of cloud computing [1]. By moving computation closer to data sources, edge architectures promise reduced latency, decreased bandwidth costs, and improved privacy guarantees. However, these benefits come at a cost: edge nodes are resource-constrained, heterogeneous, and geographically dispersed, making efficient resource allocation a fundamentally harder problem than in centralized cloud environments.

Consider a representative deployment scenario: a metropolitan 5G network operator deploys 64 edge nodes across 12 locations to serve video analytics, IoT stream processing, and inference workloads from multiple tenants. Each tenant specifies latency SLAs, throughput requirements, and isolation constraints. The operator's objective is to maximize aggregate utilization while satisfying all tenant contracts — a problem we call *multi-tenant edge-aware placement* (MTEAP).

Key Insight

Existing schedulers treat edge nodes as interchangeable commodity units. In practice, edge heterogeneity — differences in GPU availability, memory bandwidth, and network topology — creates placement asymmetries that, when exploited correctly, unlock substantial efficiency gains.

1.2 Thesis Contributions

This thesis makes the following contributions:

1. **EdgeSched Framework.** We design and implement a modular scheduling architecture that separates workload characterization, node profiling, and placement opti-

mization into independent, composable stages.

2. **Competitive Analysis.** We provide the first formal competitive analysis of multi-tenant edge scheduling, proving an $O(\log n)$ competitive ratio for the EdgeSched online algorithm under adversarial workload sequences.

3. **Colocation-Aware Packing.** We introduce *interference affinity matrices*, a lightweight method for predicting performance degradation when workloads share compute and memory resources on a single edge node.

4. **System Evaluation.** We evaluate EdgeSched on a 64-node physical testbed using production-derived workload traces, demonstrating significant improvements over Kubernetes default scheduler, FIRMAMENT [2], and Poseidon [3].

1.3 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 surveys background and related work in scheduling, edge computing, and resource allocation. Chapter 3 develops the formal model for MTEAP and proves key structural properties. Chapter 4 presents the EdgeSched architecture and algorithms in detail. Chapter 5 describes the evaluation methodology and results. Chapter 6 concludes with a summary of contributions and directions for future work.

2. Background and Related Work

2.1 Edge Computing Landscape

Edge computing sits between the device layer (sensors, mobile phones, IoT endpoints) and the cloud layer (centralized data centers). Edge nodes in a production environment are heterogeneous: a roadside unit might have a single x86 CPU and no GPU, while an edge micro-datacenter might feature multiple GPUs and terabytes of NVMe storage. Designing schedulers that work across this heterogeneity spectrum is the central challenge addressed in this thesis.

2.2 Resource Allocation in Distributed Systems

Table 2.1: Comparison of resource allocation strategies across deployment tiers.

Strategy	**Cloud**	**Edge**	**This Work**
Scheduling Granularity	VM / Container	Pod / Function	Pod + Colocation Group
Heterogeneity Handling	None assumed	Ad-hoc	First-class, profiled
Latency Target	$<$100 ms	$<$10 ms	$<$1 ms scheduling overhead
Colocation Model	Bin-packing	Not modeled	Interference affinity matrix

Table 2.1 positions this work relative to cloud and existing edge allocation strategies. The key differentiator is the *interference affinity matrix*, which we introduce in Chapter 3.

2.3 Limitations of Prior Approaches

Kubernetes, the dominant container orchestration platform, provides a default scheduler that scores nodes via a weighted sum of resource availability predicates. This approach struggles with edge deployments for three reasons: it assumes homogeneous nodes, it

ignores workload colocation effects, and its scheduling overhead becomes significant at the sub-millisecond latency targets required by edge applications.

Limitation

The Kubernetes default scheduler makes placement decisions node-by-node, evaluating every candidate. For a cluster with n nodes and m pending pods, this yields $O(nm)$ complexity — acceptable in cloud datacenters, prohibitive at edge scale where n can reach thousands across metro deployments.

Recent academic schedulers — FIRMAMENT [2], YARN [4], and Apollo [5] — address some of these issues but still operate under cloud-scale assumptions. None model multi-tenant isolation as a first-class constraint, and none account for interference between colocated workloads on resource-constrained edge hardware.

3. Formal Model

3.1 System Model

We model an edge cluster as a set of M heterogeneous nodes $\mathcal{N} = \{n_1, n_2, \ldots, n_M\}$, each characterized by a capacity vector $\mathbf{c}_i \in \mathbb{R}^d$ where d denotes the number of resource dimensions (CPU cores, memory GiB, GPU fraction, network bandwidth Mbps).

A workload W is defined by a tuple $(\mathbf{r}, \ell, \tau, \sigma)$ where:

- $\mathbf{r} \in \mathbb{R}^d$ is the resource demand vector.
- ℓ is the maximum tolerable latency (SLA), in milliseconds.
- τ is the workload type (video analytics, stream processing, inference, batch).
- σ is the estimated throughput, in requests per second.

3.2 Interference Affinity Matrices

A core contribution of this work is the *interference affinity matrix* (IAM), which quantifies the performance degradation experienced when two workloads are colocated on the same node.

For workload types τ_i, τ_j, the affinity coefficient $a_{ij} \in [0, 1]$ represents the fraction of peak throughput retained when both run on a single node:

$$a_{ij} = \frac{\text{Throughput}_{\tau_i}^{\text{colocated}}}{\text{Throughput}_{\tau_i}^{\text{dedicated}}}$$

The IAM values in Table 3.1 reveal a key insight: **inference workloads** suffer the most from colocation, particularly when paired with batch processing (affinity 0.52). This finding motivates dedicated inference partitions, a design choice we implement in Chapter 4.

Table 3.1: Measured interference affinity coefficients (mean across 64 edge nodes). Lower values indicate stronger interference.

	Video	**Stream**	**Inference**	**Batch**
Video Analytics	0.92	0.71	0.63	0.88
Stream Processing	0.71	0.95	0.58	0.91
Inference Serving	0.63	0.58	0.89	0.52
Batch Processing	0.88	0.91	0.52	0.96

4. EdgeSched Architecture

4.1 System Overview

EdgeSched is organized as a pipeline of three stages:

Stage 1: Profiling

Lightweight agents on each edge node periodically measure resource availability (CPU, memory, GPU, network) and report a compact **NodeCapability** vector to the scheduler. This profiling runs with $<0.1\%$ CPU overhead on production nodes.

Stage 2: Workload Classification

Incoming workloads are classified by type using a lightweight classifier trained on historical traces. The classifier outputs a resource demand vector $\mathbf{r}$ and a type label τ used for IAM lookup.

Stage 3: Placement Optimization

A constraint-solving layer assigns workloads to nodes, maximizing utilization while respecting capacity limits, SLA constraints, and IAM-based colocation penalties. The solver uses an online greedy algorithm with worst-case guarantee $O(\log n)$.

4.2 Placement Algorithm

Algorithm 4.2 describes the core online scheduling loop.

Algorithm 1: EdgeSched Online Placement

Input: Pending workload $W = (\mathbf{r}, \ell, \tau, \sigma)$

Input: Node set $\mathcal{N}$ with current allocations $\mathcal{A}$

Output: Target node n^* or reject

1. Candidate set $\mathcal{C} \leftarrow \{n \in \mathcal{N} \mid \mathbf{c}_n - \sum_{w \in \mathcal{A}_n} \mathbf{r}_w \geq \mathbf{r}\}$
2. **if** $\mathcal{C} = \emptyset$ **then return** reject
3. For each $n \in \mathcal{C}$, compute score $s_n = \text{util}(n) + \lambda \cdot \text{colocatePenalty}(n, W)$
4. Select $n^* = \arg\min_{n \in \mathcal{C}} s_n$
5. **return** n^*

The parameter $\lambda \in [0, 1]$ trades off utilization against colocation penalty. Higher λ favors dedicated nodes for sensitive workloads.

The colocation penalty function uses the IAM:

$$\text{colocatePenalty}(n, W) = \frac{1}{|\mathcal{A}_n|} \sum_{w \in \mathcal{A}_n} (1 - a_{\tau_w, \tau})$$

This formulation penalizes nodes that already host workloads with high interference affinity against the incoming workload — effectively implementing soft anti-affinity without the rigidity of hard constraints.

5. Evaluation

5.1 Experimental Setup

5.1.1 Hardware Configuration

The testbed consists of 64 heterogeneous edge nodes distributed across four racks, summarized in Table 5.1. All nodes run Ubuntu 22.04 LTS with kernel 6.5 and Docker 24.0.

Table 5.1: Edge cluster node configuration.

Tier	Count	CPU	Memory
Standard	36	8-core x86	32 GB
GPU-Enhanced	16	12-core + Lumina T4	64 GB
High-Memory	12	16-core	128 GB

5.1.2 Workload Traces

We use three workload profiles derived from production traces at Vertex Cloud Services, sanitized and released for academic use:

- **Video Analytics Pipeline.** 480p and 720p streams processed through object detection (YOLO-style) and frame differencing. Target latency: 50 ms per frame.

- **IoT Stream Aggregation.** Time-series window aggregation (median, percentile, anomaly score) over sensor streams at ingress rates up to 10 000 events per second per node.

- **Inference Serving.** Bert-medium and ResNet-50 serving behind a load balancer, with Poisson arrival at mean 200 queries per second.

5.2 Results

5.2.1 Tail Latency Reduction

Figure **??** would show the cumulative distribution of end-to-end request latency across all workloads. EdgeSched reduces the 99th-percentile latency by 42% compared to the Kubernetes default scheduler and by 27% compared to FIRMAMENT.

5.2.2 Node Utilization

Table 5.2: Aggregate node utilization across schedulers (mean over 24-hour traces).

Scheduler	CPU Util.	Mem Util.	GPU Util.
Kubernetes Default	47.2%	52.8%	31.4%
FIRMAMENT [2]	58.9%	61.5%	44.7%
EdgeSched	**73.4%**	**78.1%**	**67.2%**

Table 5.2 confirms that EdgeSched achieves the highest utilization across all resource dimensions. The GPU utilization improvement is particularly significant — a 2.14× increase over the Kubernetes baseline — because the IAM-aware placement avoids co-locating GPU-sensitive workloads.

Statistical Significance

All reported improvements are statistically significant at $p < 0.01$ using a two-tailed Student's t-test over 10 independent 24-hour trial runs, each with randomized workload arrival order.

6. Conclusion and Future Work

6.1 Summary of Contributions

This thesis introduced EdgeSched, a resource allocation framework for multi-tenant edge computing clusters that addresses the fundamental tension between utilization, latency, and isolation. We formalized the multi-tenant edge-aware placement problem (MTEAP), introduced the interference affinity matrix as a lightweight mechanism for modeling colocation effects, and provided competitive analysis proving an $O(\log n)$ guarantee for the online scheduling algorithm. Evaluation on a 64-node physical testbed demonstrated tail-latency reductions of up to 42% and utilization improvements of up to 31% over state-of-the-art alternatives.

6.2 Future Directions

Several avenues remain open for investigation:

1. **Learned IAM.** The interference affinity matrix is currently profiled offline. Online learning of IAM coefficients from production traces would enable adaptation to workload drift and novel workload types.

2. **Federated Edge Scheduling.** This work assumes a single scheduling domain. Hierarchical and federated scheduling across multiple edge domains (e.g., across cell towers owned by different operators) requires distributed consensus mechanisms.

3. **Integration with Serverless.** Serverless edge platforms such as Cloudflare Workers and AWS Lambda@Edge introduce cold-start and invocation-pattern considerations that the current model does not capture.

4. **Sustainability-Aware Placement.** Incorporating carbon intensity signals into scheduling decisions would allow edge operators to shift workloads temporally and spatially toward cleaner energy sources.

The journey from cloud to edge is accelerating. Efficient scheduling is the quiet infrastructure that determines whether that journey succeeds. This thesis offers one piece of that foundation.

Bibliography

[1] K. Talwar, M. Patel, and R. Iyer, "Edge computing: A taxonomy and survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3496–3537, 2019.

[2] I. Gog, M. Schwarzkopf, A. Gleave, R. N. M. Watson, and S. Hand, "Firmament: Fast, centralized cluster scheduling at scale," in *Proc. USENIX OSDI*, 2016, pp. 99–115.

[3] A. Gupta, N. Sharma, and P. Kumar, "Poseidon: Latency-aware scheduling for geo-distributed edge clusters," in *Proc. ACM SoCC*, 2021, pp. 201–215.

[4] V. K. Vavilapalli *et al.*, "Apache Hadoop YARN: Yet another resource negotiator," in *Proc. ACM SoCC*, 2013, pp. 1–16.

[5] E. Boutin *et al.*, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *Proc. USENIX OSDI*, 2014, pp. 285–300.