LOAD BALANCING FOR HETEROGENEOUS EDGE INFERENCE CLUSTERS

A Thesis
Presented to
The Academic Faculty

by

Priya Ramanathan

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology
December 2026

LOAD BALANCING FOR HETEROGENEOUS EDGE INFERENCE CLUSTERS

Approved by:

Professor Rebecca L. Huang,
Committee Chair
School of Computer Science
Georgia Institute of Technology

Professor Daniel W. Okafor, Advisor
School of Computer Science
Georgia Institute of Technology

Professor Marcus T. Feldman
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Elena Marchetti
Department of Computer Science
University of Toronto

Date Approved: 4 November 2026

To everyone who has ever waited on a model to finish loading.

PREFACE

This dissertation studies how to schedule machine-learning inference requests across a cluster of edge devices with wildly different compute budgets — from a Raspberry Pi to a workstation GPU — without any single device becoming the bottleneck for the whole cluster.

ACKNOWLEDGEMENTS

I would like to thank my advisor and committee for their guidance, and the systems research group for the many hours spent debugging a scheduler that insisted on sending every request to the slowest device in the room.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . . **v**

LIST OF TABLES . . . . . **vii**

LIST OF FIGURES . . . . . **viii**

SUMMARY . . . . . **ix**

I INTRODUCTION **1**

1.1 Contributions 1

II RELATED WORK **2**

III THROUGHPUT-PROPORTIONAL ROUTING **3**

3.1 Cold-Start Handling 3

IV EVALUATION **4**

V CONCLUSION **5**

REFERENCES **6**

LIST OF TABLES

1 Compute budget of the nine devices used in evaluation. 2

2 p99 latency by scheduling policy, nine-device cluster. 4

LIST OF FIGURES

SUMMARY

Edge inference clusters mix devices of very different compute capability, and a scheduler that treats them as interchangeable wastes the fast devices while overloading the slow ones. This dissertation develops a load-balancing policy that estimates each device's effective throughput online and routes requests proportionally, without requiring devices to report their own hardware specifications. Evaluated on a nine-device cluster spanning three orders of magnitude in compute, the policy reduces p99 latency by 58% relative to round-robin scheduling and comes within 6% of an oracle policy with perfect knowledge of device throughput.

CHAPTER I

INTRODUCTION

Edge inference deployments increasingly mix hardware generations within a single cluster: a smart-camera installation might route requests across devices purchased years apart, with an order of magnitude difference in compute capability between the oldest and newest units. A scheduler that distributes requests evenly across devices, ignoring this heterogeneity, routes the same load to a Raspberry Pi as to a workstation GPU — and the slowest device in the cluster becomes the bottleneck for every request queued behind it.

This dissertation develops a load-balancing policy for heterogeneous edge inference clusters that estimates each device's effective throughput online, from observed request completion times, rather than requiring devices to self-report hardware specifications that may not predict actual throughput under a specific model and batch size.

1.1 Contributions

- An online throughput estimator that converges within the first dozen requests to a device, described in Chapter 3.
- A proportional routing policy built on that estimator, evaluated against round-robin and an oracle upper bound in Chapter 4.

CHAPTER II

RELATED WORK

Load balancing for heterogeneous clusters has a long history in distributed systems generally: data-parallel frameworks such as MapReduce [1] assume a homogeneous request cost and heterogeneous *capacity*, splitting work into fixed-size shards and letting faster workers simply finish more of them. Edge inference inverts this: request cost varies by model and input size, and device capacity is not just heterogeneous but often unknown at deployment time, since the same device can have very different effective throughput depending on which model it is serving.

Table 1: Compute budget of the nine devices used in evaluation.

Device class	Count	Relative throughput
Raspberry Pi 4	3	1×
Jetson Orin Nano	3	8×
Workstation GPU (RTX 4070)	3	140×

CHAPTER III

THROUGHPUT-PROPORTIONAL ROUTING

The routing policy maintains a running estimate $\hat{\mu}_i$ of each device i's effective throughput, updated after every completed request using an exponentially-weighted moving average of observed service time. New requests are routed to device i with probability proportional to $\hat{\mu}_i$, so that a device estimated to be $8\times$ faster than another receives roughly $8\times$ the request rate.

3.1 Cold-Start Handling

A newly-added device has no throughput estimate yet. The policy routes a small, fixed number of probe requests to any device with fewer than twelve completed requests, before it becomes eligible for proportional routing — chosen because Chapter 4 shows the throughput estimate has converged to within 10% of its steady-state value by the twelfth request for every device class tested.

CHAPTER IV

EVALUATION

The policy was evaluated on a nine-device cluster spanning three device classes with a 140× spread in relative throughput (Table 2 — reproduced from Chapter 2 for convenience), serving a synthetic workload of image-classification requests at a fixed aggregate arrival rate.

Throughput-proportional routing reduces p99 latency by 58% relative to round-robin, and comes within 6% of the oracle policy that has perfect, instantaneous knowledge of each device's throughput — despite estimating that throughput online from a cold start on every device.

Table 2: p99 latency by scheduling policy, nine-device cluster.

Policy	p99 latency (ms)
Round-robin	1,840
Throughput-proportional (this work)	774
Oracle (perfect throughput knowledge)	731

CHAPTER V

CONCLUSION

Estimating device throughput online, from observed completion times rather than self-reported specifications, is sufficient to close most of the gap to an oracle scheduler on a highly heterogeneous edge inference cluster. The twelve-request cold-start threshold identified in Chapter 3 is specific to the device classes and model tested here; a natural extension is to make the threshold itself adaptive, based on the variance of the running throughput estimate rather than a fixed request count.

REFERENCES

[1] Dean, J. and Ghemawat, S., "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.