Malcolm: Multi-agent Learning for Cooperative Load Management at Rack Scale

Ali Hossein Abbasi Abyaneh
University of Waterloo
Waterloo, Canada
a36hosse@uwaterloo.ca

Maizi Liao
University of Waterloo
Waterloo, Canada
m7liao@uwaterloo.ca

Seyed Majid Zahedi
University of Waterloo
Waterloo, Canada
smzahedi@uwaterloo.ca

ABSTRACT

We consider the problem of balancing the load among servers in dense racks for microsecond-scale workloads. To balance the load in such settings, tens of millions of scheduling decisions have to be made per second. Achieving this throughput while providing microsecond-scale latency is extremely challenging. To address this challenge, we design a fully decentralized load-balancing framework, which allows servers to collectively balance the load in the system. We model the interactions among servers as a cooperative stochastic game. To find the game's parametric Nash equilibrium, we design and implement a decentralized algorithm based on multi-agent-learning theory. We empirically show that our proposed algorithm is adaptive and scalable while outperforming state-of-the art alternatives. The full paper of this abstract can be found at https://doi.org/10.1145/3570611.

ACM Reference Format:
Ali Hossein Abbasi Abyaneh, Maizi Liao, and Seyed Majid Zahedi. 2023. Malcolm: Multi-agent Learning for Cooperative Load Management at Rack Scale. In *Abstract Proceedings of the 2023 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '23 Abstracts), June 19–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3578338.3593550

1 INTRODUCTION

To process user requests, popular datacenter applications such as web search, e-commerce, and social networks rely on responses from thousands of services. In such applications, end-to-end response times are dictated by the slowest response [3]. To guarantee fast responses, datacenter services are governed by strict service-level objectives (SLOs). To meet these SLOs, it is imperative to provide high throughput at microsecond-scale latency [1].

In recent years, there has been significant work on designing microsecond-scale schedulers for multi-core servers (e.g., [2, 7]). While these solutions achieve microsecond-scale tail latencies for multi-core servers, they do not scale beyond a few tens of cores. A typical high-density datacenter rack can comprise thousands of interconnected, heterogeneous computing units. The increasing rack density poses new challenges for designing rack-scale schedulers. To address these challenges, the state of the art proposes a two-layer hierarchical scheduler consisting of a high-level inter-server scheduler and low-level intra-server schedulers [9]. Each intra-server scheduler balances the load between cores in a server, and the inter-server scheduler balances the load between servers.

To realize centralized rack-scale scheduling, the inter-server scheduler is implemented in programmable ToR switches. The key benefit of this approach is that the ToR switch can schedule tasks at the line rate as it already is on the path of all tasks sent to the rack. However, this approach has three main limitations. First, it requires a programmable switch, which limits its deployment in datacenters without programmable switches. Second, it imposes additional functionality to the packet switching fabric, ultimately leading to degraded network throughput [6]. Third, due to restricted resources available on a programmable switch, power-of-2-choices policy is used to approximate cFCFS, which can perform very poorly in the presence of heterogeneity [8].

To address the limitations of the state of the art, we present Malcolm, a heterogeneity-aware load-balancing framework that allows servers in the rack to collectively balance the load between themselves. We model the interactions among servers as a cooperative stochastic game, and use robust, game-theoretic analysis to study load-balancing strategies. Furthermore, to find the game's parametric Nash equilibrium, we design and implement a decentralized multi-agent learning algorithm. In our proposed solution, servers make scheduling decisions in tens of nanoseconds based on (possibly out-of-date) estimates of the load on other servers. Our implementation allows decentralized coordination among servers through infrequent network communications.

2 MALCOLM ARCHITECTURE

Malcolm consists of heterogeneous servers interconnected by a high-bandwidth, low-latency network fabric in a rack. Clients send their tasks to servers at different rates. Each server runs one or more *Malcolm nodes*. Malcolm uses a multi-layer approach with intra-node and inter-node schedulers. Each Malcolm node consists of a *centralized task scheduler* for intra-node scheduling and a *load manager* for inter-node load balancing.

Centralized intra-node schedulers implement an FCFS scheduler to schedule tasks among available worker threads within each Malcolm node. Malcolm adopts a distributed inter-node scheduling approach. Inter-node load managers cooperatively balance the load among Malcolm nodes at per-task granularity. Each Malcolm node also consists of a *heartbeat manager*, which dynamically learns load-balancing strategies by interacting with other nodes. Nodes regularly send heartbeat messages to each other.

Upon receiving a new task, the load manager decides whether to accept the task or migrate it to another node. This decision is

SIGMETRICS '23 Abstracts, June 19–23, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0074-3/23/06.
https://doi.org/10.1145/3578338.3593550

made based on nodes' (possibly out-of-date) loads. After processing each task by a worker thread, the load manager decides if it needs to steal tasks from other nodes. Work-stealing decisions on task completions complement the migration decisions on task arrivals. Malcolm uses game theory to model rack dynamics and load balancing decisions.

3 DISTRIBUTED LOAD-BALANCING GAME

We present the distributed load-balancing game (DLB) used in Malcolm as a framework to balance the load between nodes at rack scale. The game consists of N heterogeneous nodes, represented by N agents. Different agents can receive and process tasks at different rates. Upon receiving a task, each agent decides whether to keep the task or migrate it to another agent. Upon completing a task, each agent decides if it needs to steal a task from another agent. The state of the game evolves over time as scheduling decisions collectively shape the load on different nodes. The goal of each agent is to minimize the total load imbalance in the rack while migrating and stealing the minimum number of tasks.

We model the DLB game as a team Markov game. To allow for practical solutions, we focus on parametric strategies. Agents optimize their parametric policies to maximize their expected long-term payoff. A *Nash equilibrium* (NE) is a strategy profile in which all agents simultaneously play best responses against each others' strategies. In general, the problem of finding an NE is computationally expensive. Fortunately, an NE could be computed in polynomial time for the DLB game. This is because the DLB game is a *Markov potential game (MPG)*. And for an MPG, an NE can be obtained in polynomial time by solving a Markov decision process [5].

To find a parametric NE of the DLB game, we propose a multi-agent-learning algorithm inspired by recent advances in decentralized machine learning techniques [4]. The algorithm is model free and does not require any prior knowledge of system dynamics. Moreover, the algorithm is guaranteed to find parametric NE strategies of the DLB game in a distributed manner and can be implemented and deployed in practice.

4 IMPLEMENTATION

To provide low-latency node-to-node communication, Malcolm uses a user-space networking stack. For intra-node task scheduling, Malcolm implements the FCFS policy by using a centralized lock-free task queue that is shared by all worker threads. Malcolm's design allows recent dataplane operating systems and server-level schedulers such as ZygOS [7] to be deployed for scheduling tasks between worker threads. For inter-node load balancing and node-to-client communications, Malcolm node has a dedicated load-manager thread. The load-manager thread is responsible for handling incoming tasks and sending responses back to clients.

On the arrival of each task, the load manager consults the migration policy to decide whether to accept the task or migrate it to another node. Since migration policy is probabilistic, consulting the policy involves generating a random number uniformly from 0 to 1. Based on measurements on our testbed, this takes only tens of nanoseconds. Once each task is completed, the load manager consults the work-stealing policy to decide whether to send a work-stealing request to other servers. Consulting the work-stealing policy is similar to consulting the migration policy.

Malcolm dedicates a heartbeat-manager thread to update policy parameters and maintain migration and work-stealing probabilities. The heartbeat manager also broadcasts heartbeat messages to other nodes at fixed *heartbeat intervals*. Each heartbeat interval provides a new data point (i.e., the old state, taken actions, and the new state). Each Malcolm node collects data points to form a training dataset for updating policy parameters (i.e., actor and critic parameters). To handle microsecond-scale workloads, the execution time of a policy update has to be at most several microseconds. Unfortunately, implementing the algorithm using off-the-shelf machine-learning frameworks, such as PyTorch, falls short of meeting this requirement. To achieve high performance, in Malcolm, we use linear function approximation and derive closed-form formulas for all parameter updates. We implement these closed-form formulas in about 600 lines of C++ code. To speed up vector-to-vector multiplications, our implementation uses x86 AVX2 instructions, which are available in most datacenter servers. Our implementation offers up to 300× speedup in execution time compared to a PyTorch implementation. The code of Malcolm is open-source and available at https://github.com/uwaterloo-mast/malcolm.

5 EVALUATION

We use a diverse set of synthetic benchmarks to evaluate the performance, scalability, and adaptivity of Malcolm on homogeneous and heterogeneous rack configurations. We compared Malcolm's performance to other state-of-the-art baselines. In homogeneous settings, Malcolm performs as well as the best alternative among other baselines. In heterogeneous settings, compared to other baselines, for lower loads, Malcolm improves tail latency by up to a factor of four. And for the same tail latency, Malcolm achieves up to 60% more throughput compared to the best alternative among other baselines.

REFERENCES

[1] Luiz Barroso, Mike Marty, David Patterson, and Parthasarathy Ranganathan. 2017. Attack of the Killer Microseconds. *Communications of the ACM (CACM)* 60, 4 (mar 2017), 48–54.

[2] Adam Belay, George Prekas, Mia Primorac, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. 2016. The IX operating system: Combining low latency, high throughput, and efficiency in a protected dataplane. *ACM Transactions on Computer Systems (TOCS)* 34, 4 (2016), 1–39.

[3] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Communications of the ACM (CACM)* 56, 2 (2013), 74–80.

[4] Donghwan Lee, Niao He, Parameswaran Kamalaruban, and Volkan Cevher. 2020. Optimization for reinforcement learning: From a single agent to cooperative agents. *IEEE Signal Processing Magazine* 37, 3 (2020), 123–135.

[5] Sergio Valcarcel Macua, Javier Zazo, and Santiago Zazo. 2018. Learning Parametric Closed-Loop Policies for Markov Potential Games. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[6] James McCauley, Aurojit Panda, Arvind Krishnamurthy, and Scott Shenker. 2019. Thoughts on Load Distribution and the Role of Programmable Switches. *ACM SIGCOMM Computer Communication Review* 49, 1 (2019), 18–23.

[7] George Prekas, Marios Kogias, and Edouard Bugnion. 2017. ZygOS: Achieving low tail latency for microsecond-scale networked tasks. In *Proceedings of the 26th Symposium on Operating Systems Principles, (SOSP)*. 325–341.

[8] Alexander L Stolyar. 2015. Pull-based load distribution in large-scale heterogeneous service systems. *Queueing Systems* 80, 4 (2015), 341–361.

[9] Hang Zhu, Kostis Kaffes, Zixu Chen, Zhenming Liu, Christos Kozyrakis, Ion Stoica, and Xin Jin. 2020. RackSched: A microsecond-scale scheduler for rack-scale computers. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 1225–1240.