Optimizing Distributed Systems using Machine Learning

Ignacio A. Cano

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Arvind Krishnamurthy, Chair

Xi Wang

Kevin Jamieson

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science & Engineering

University of Washington

Abstract

Optimizing Distributed Systems using Machine Learning

Ignacio A. Cano

Chair of the Supervisory Committee:
Professor Arvind Krishnamurthy
Paul G. Allen School of Computer Science & Engineering

Distributed systems consist of many components that interact with each other to perform certain task(s). Traditionally, many of these systems base their decisions on sets of rules or configurations defined by operators as well as handcrafted analytical models. However, creating those rules or engineering such models is a challenging task. First, the same system should be able to work under a combinatorial number of conditions on top of heterogeneous hardware. Second, they should support different type of workloads and run in potentially widely different settings. Third, they should be able to handle time-varying resource needs. These factors render reasoning about distributed systems' performance in general far from trivial.

In this thesis, we propose optimizing distributed systems using machine learning (ML). Our main contribution is the design, implementation, augmentation, and evaluation of three distributed systems that illustrate the impact of these ML-based optimizations: 1) CURATOR, a framework that safeguards distributed storage systems' health and performance by scheduling and executing background maintenance tasks, 2) ADARES, an adaptive system that dynamically adjusts virtual machine resources in virtual execution environments, and 3) PULPO, a federated system that efficiently trains machine learning models across different data centers. Each system instantiates appropriate ML models for the task at hand, alleviating systems designers from manually tuning rules and handcrafting complex analytical models. Our evaluations on real clusters show how our ML formulations result in improved distributed systems' efficiency and performance.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Many people have helped me throughout this long journey. First and foremost, I would like to thank my advisor, Arvind Krishnamurthy. I wouldn't be writing this thesis if it weren't for his always wise guidance and advise. It has been an honor and a tremendous pleasure to work with him, not only an excellent researcher but also (and most importantly) an excellent person.

I would like to thank the other members of my committee, Xi Wang, Kevin Jamieson, and Radha Poovendran. Their always precise and constructive feedback have helped us raise the bar and improve the projects that compose this thesis.

Throughout the program I was able to intern at three great companies, Microsoft, Nutanix, and Google. I would like to particularly thank my awesome mentors Markus Weimer, Srinivas Aiyar, and Andrey Gubichev. Also, thanks to my collaborators Dhruv Mahajan, Carlo Curino, Chern Cheah, Karan Gupta, Petros Venetis, and Pedro Fonseca, for all their help throughout our projects.

I have been truly lucky to meet and interact with so many brilliant people here at the University of Washington. I would like to thank my closest UW friends and office mates, Shrainik Jain and Vincent Lee. We have shared the ups and downs of grad school over the past 5+ years, and we are finally seeing the light at the end of the tunnel. Thanks to the rest of the gang, Shumo Chu, Alex Mariakakis, Koosha Khalvati, and Aleksander Holynski, who have made my tenure here really enjoyable. I am also grateful to Ming Liu, Danyang Zhuo, Naveen Kr. Sharma, Anna Simpson, Lequn Chen, and Antoine Kaufmann for all their valuable feedback.

I would also like to thank Tianqi Chen and Marco Tulio Ribeiro, my former lab mates. Thanks to Sameer Singh, Ben Taskar, and Carlos Guestrin for supporting and advising me during the beginnings. I have been really lucky to meet (and briefly work) with Ben before his unexpected passing.

I am also thankful to Elise DeGoede for all her support and help throughout the program. Thanks to my sponsors, the Argentinean Government and Fulbright, for their financial support during my first two years. I also want to thank the other Argentinean fellows, Bruce, Pipe, Marca, Colitos, Albert, Eze, and Chino, who have made this great experience even better. Special thanks to Nico and Kaddy, our family here in Seattle, and to all my other friends, here and back home, for all the great memories.

Finally, I would like to thank my amazing wife, Laura, for joining me in this great adventure and for always being there. I have been blessed with two incredible daughters, Magda and Coco. Raising and watching them grow during the course of my studies has been, by far, the most rewarding experience of all. Last but not least, I would like to thank all my family and in-laws for supporting us from Argentina. Special thanks to my parents, Carmen and Ernesto, for always encouraging me to pursue my dreams and inculcating me the two core values that were fundamental to bring this effort to an end: perseverance and hard work.

Dedication

to my dear wife, Laura

to my beloved daughters, Magda and Coco

to my wonderful parents, Carmen and Ernesto

1 | Introduction

Distributed systems comprise of many components that interact and cooperate with each other to perform certain task(s). For example, storage tiering services automatically migrate data between solid-state (SSDs) and hard disk drives (HDDs) based on usage, virtualization software allows to package applications on virtual machines (VMs) and execute them together with other workloads on the same physical hardware, geographically distributed systems support data pipelines to join and analyze disperse datasets, and so on.

Many of these real-world systems are heuristic-based; that is, they heavily rely on (user-defined) policies, rules, or configurations derived from specific domain knowledge in order to make decisions, and they typically use analytical models to describe their behavior. However, the design of these systems, together with their rules and models, is an inherently difficult task. In this regard, the limit is usually the complexity that arise from the large number of components and connections, the intricate component dependencies, the irregular interactions and resource needs, and the imprecise descriptions. Further, the same system might need to handle different workloads in different settings, as well as run in heterogeneous hardware platforms or even across different geographic regions. To make matters worse, these workloads might also change over time and applications may interfere with each other while contending for shared resources.

These factors make it hard to reason about the performance of distributed systems in general, complicate the ability to create accurate models, and render static configurations inappropriate. Even though many highly qualified engineers are typically involved in performance improvement tasks, we still often see (surprisingly) low efficiency in many of these systems in production.

In order to overcome these difficulties, it becomes necessary to build smarter distributed systems that could learn and improve their performance over time. By leveraging machine learning techniques, this new generation of self-tuning learning systems would alleviate systems designers from the task of manually manipulating knobs, tuning complicated rules or policies, and creating complex mechanisms based on handcrafted analytical models, thus significantly reducing human involvement in modern software development processes. Such systems would, instead, be able to learn the underlying complex dynamics where they operate in a way that would better resemble reality, and would generalize and adapt better, on a case-by-case basis, at runtime, which in turn would foster (significant) efficiency and performance boosts.

1.1 The Machine Learning Revolution

Machine learning is the sub-field within Artificial Intelligence (AI) that deals with the study of algorithms that improve their performance at some task with experience. More precisely, a computer program *learns* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [150].

Over the last decades, the impressive growth in data and computational capabilities have enabled the use of machine learning in many fields, from healthcare and medicine, passing through manufacturing and retail, to self-driving cars and financial services. Almost every recent technological advance involves some aspect of machine learning. Computers powered by machine learning can today perform many tasks that not long ago were assumed to require human cognition [39].

Gartner estimates that by 2022, one in five workers engaged in mostly non-routine tasks will rely on some form of AI to do a job. In 2021, AI augmentation (i.e., a combination of human and artificial intelligence, where both complement each other) will generate \$2.9 trillion in business value and recover 6.2 billion hours of worker productivity [86]. Within AI, machine learning attracted around 60% of the estimated \$8-12 billion in investments during 2016 [39]. It comes as no surprise that machine learning is and will continue to be a fundamental player in digital strategies all over the world.

1.2 Machine Learning for Distributed Systems

Distributed systems are everywhere and we are interested in their efficiency. Even though the advent of virtualization led to significant improvements in terms of the utilization of computational resources, recent studies show that computational usage levels of distributed systems deployed in private and public clouds are still far from optimal [53, 43, 63, 226]. Even more, studies show that communication inefficiencies also lead to problems [80, 95, 218]. We therefore need new ways to make computers more capable [188, 97, 180].

In our quest for finding new ways of making distributed systems more efficient, we first need to take a step back and identify the sources of the problem. Multiple reasons come to mind when we want to understand why many contemporary systems are still inefficient. Despite the evolution of workload and application demands, most systems still (heavily) rely on hardcoded policies that dictate their behavior, or fixed configurations, or complex built-in mechanisms that sometimes disregard the underlying problem structure nor leverage already-collected data. For example, distributed storage systems trigger expensive background maintenance tasks (e.g., garbage collection) disregarding the end-user foreground request patterns. Or users statically configure VMs resources (e.g., memory) based on (guessed) workload peak utilization metrics, but they ignore the fact that utilization levels usually present important temporal patterns. Or distributed data pipelines systems transfer huge amounts of data across data centers, and neglect the fact that transferring only summaries could suffice.

On that account, there is an opportunity for *optimizing distributed systems using machine learning*, with potentially broad implications for computing as a whole [91]. In this thesis, we leverage *problem structure* and *data* collected regularly by current systems in order to make them more efficient and responsive to different runtime conditions. To that end, we use existing machine learning techniques, both to inform decisions as well as to drive system behavior.

1.3 Machine Learning Modeling for Distributed Systems

The existing literature on machine learning is extensive and encompasses different areas, ranging from optimization theory to algorithms and applications. The suitability of the actual underlying machine learning model used to optimize a distributed system mainly depends on the characteristics and constraints of the problem domain.

In particular, we identify three broad classes of techniques that help increasing distributed systems' efficiency and performance, along different dimensions.

Whenever we have access to a training set of labeled examples and we are interested in learning rules that can predict labels of future unseen examples, we can rely on batch learning (also called statistical learning), probably the most common supervised learning setting [61]. Learning such rules is typically done through an iterative procedure using mathematical optimization techniques. However, the scale of modern datasets and/or the application constraints sometimes require systems to perform such optimization in a distributed fashion. In this distributed setting, a critical challenge is to efficiently communicate and coordinate information across machines [191]. By *co-designing* systems with machine learning through the lens of optimization, we could potentially improve both system's computational efficiency as well as reduce communication overheads.

On the other hand, if the system needs to adapt to changes, or even if it does not have (enough) training data to begin with, we can resort to online learning. The online learning framework is a powerful way of dealing with sequential decision problems, and the algorithms are typically fast, memory-efficient, and simple to implement [62]. An online learning algorithm receives a sequence of examples and processes them one-by-one. On each round, the algorithm receives an example and predicts its label. Then, the algorithm receives the (correct) label corresponding to the example, and uses this new labeled example to improve its predictions on subsequent rounds [61]. This generic framework can be applied to build highly adaptive distributed systems, where their behavior could be (mainly) driven by online models that evolve over time. However, as these systems would only take the (seemingly) best action suggested by the ML models, there is no way of knowing what *would have happened* had they taken a different action; that is, maybe a different action could have been better. By leveraging *bandit-based* algorithms [38], which fall under the online learning umbrella, these systems would, by design, explore different actions so as to refine their estimates, for which they would then receive immediate feedback, i.e., was it the "right" action to take? For example, in the case of virtual environments, where resource allocation changes to VMs can exhibit immediate performance improvements.

Finally, there are systems that face situations where they do not receive immediate feedback and where the current decisions can potentially impact performance over a long horizon. For example, in a distributed storage tiering service, moving infrequently accessed data from SSDs to HDDs may exhibit a delayed penalty after a non-expected future access. Although this setting can be seen as an extension of the previous approach, the feedback sparsity makes the problem (much) harder to learn. By leveraging *reinforcement learning-based* techniques [199, 201, 198], these systems would be able to perform tasks with non-immediate feedback, as well as adapt to changing dynamics.

All in all, there are many machine learning modeling techniques to choose from when applying ML to optimize distributed systems. As a rule of thumb, simplicity is something that should always be a key goal in the system design and unnecessary complexity should be avoided.

1.4 The Roles of Machine Learning in Distributed Systems

Besides the different machine learning modeling strategies we can use to optimize distributed systems, the control we have on the system itself is crucial in determining the role ML will play and (potentially) its impact. Applying ML to a system where we have no control over its internals is different from doing so to a system where we can change/augment its core. Similarly, applying ML to a system with years in production is different than creating a brand new system with built-in ML capabilities from the ground up.

Depending on whether we can alter or augment the policies according to which decisions are made in a distributed system, machine learning could be used as an enabler for "smarter" policies. For example, *ML-based policies* could be used to decide when to trigger background maintenance tasks in *black-box* distributed storage systems in order to reduce end-user access latency and minimize interference with foreground work. Herein, as the ML-based policy would not be that well integrated to the core of the system itself, it may experience delayed feedback on whether the decisions made were “good” or “bad”. Hence, a *reinforcement learning-based* technique might be the right modeling approach.

In the case where we have greater control, a more powerful role for ML would be to complement, augment, or perhaps replace core system mechanisms. For example, an *ML-based mechanism* could be leveraged to manage resource allocations in *gray-box* virtual execution environments, thus complementing typical memory reclamation techniques (e.g., ballooning, hypervisor swapping) and helping to multiplex the underlying physical resources. As the ML-based mechanism would be part of the system itself, it might be able to get immediate feedback of how it is doing, therefore, a *bandit-based* approach may be more appropriate.

Finally, whenever we have full control (i.e., *white-box* system) or we are at an early design stage, we could treat *ML as a first-class citizen*. In other words, we could *co-design* the system together with machine learning so that they are aware of each other: ML-aware system and system-aware ML. For example, this *ML-System co-design* could be used to efficiently trade-off computation and communication in distributed systems in order to perform cost-effective training of machine learning models across continents.

In this thesis, we present three case studies of applying machine learning to optimize distributed systems, from a simpler *ML-based policy* application, passing through an intermediate *ML-based mechanism*, to the more involved case of using *ML-System co-design*.

1.5 The Challenges of Machine Learning for Distributed Systems

Researchers are starting to leverage advances in machine learning to optimize systems, either by augmenting or replacing current heuristics and data structures [97, 119]. However, even though machine learning for systems is a very promising field, the space has only been lightly explored, arguably, due to the many challenges involved in the process. In this section, we describe some of the common challenges of using machine learning to optimize distributed systems, and how we address them in our work.

First, we need a strategy to overcome the cold start problem, especially in the bandit or reinforcement learning-based approaches, where it might take too long to distill a proper model. We want our systems to make the right decisions as soon as possible in order to minimize the costly trial and error interactions with the real environments. In this case, we propose *bootstrapping* the learning agents with already-collected historical traces as well as simulations. By this means, we leverage *transfer learning* in order to expose the agents to a broad set of relevant situations in advance, which would accelerate training and steer the systems towards more efficient states faster, thus reducing the sample complexity.

Second, we need proper setups for the machine learning models. Such setups depend on how the systems are structured in terms of what features can be collected and how the performance can be quantified. In other words, we require "cheap" mechanisms to gather sufficient training data to feed into these (oftentimes) data hungry models. To aid in the learning process, we create efficient *sensing* mechanisms as well as extend existing *logging* infrastructure in order to provide our models with a global though fine-grained view of the underlying system components. Further, we propose intuitive yet powerful functions that promote high-performance system behavior.

Third, we need an efficient way to represent high-dimensional spaces and estimate performance valuations of unseen conditions. Moreover, we need our systems to be able to make seemingly suboptimal decisions in search of better overall performance. To this end, we leverage different ML techniques to efficiently encode high-dimensional data and we rely on *regularization* mechanisms to ensure the models can generalize to real end-user workloads. Further, we foster our learning agents to *explore*, within safety bounds, different states so as to build better approximations of the true underlying dynamics.

Finally, we should not disrupt the normal functioning of the systems, i.e., we should be extra cautious and make careful decisions in order not to impair their correct behavior. Further, in the quest for real-world adoption, we need to provide insight to the operators about the models' decisions; that is, we should encourage interpretability in our models. To this end, we promote *safety* by revising ML-based decisions with manual rules to flag abnormalities and potentially discarding the machine learning-based recommendations. Moreover, we leverage models that provide *uncer-*

tainty in their predictions so as to aid the domain experts in better understanding the algorithms' decision-making process.

Overall, we follow a data-driven approach, where we use both data from real-world systems running in production and simulations in order to initialize our models. Besides leveraging existing ML algorithms, we develop the artifacts necessary for improving distributed systems' efficiency and performance, with support for heterogeneous workloads and environments, and time-varying resource needs.

1.6 Contributions

This thesis proposes using machine learning to optimize distributed systems. In order to demonstrate the effect of these ML-based optimizations, we design, implement, augment, and evaluate three real-world systems: 1) CURATOR, a framework that safeguards the health and performance of distributed storage systems by leveraging *ML-based policies* to schedule background maintenance tasks using *reinforcement learning*, 2) ADARES, an adaptive system that relies on an *ML-based mechanism* to dynamically adjust virtual machine resources running in virtual execution environments using *bandit-based techniques*, and 3) PULPO, a federation-based *system co-designed with machine learning* optimization techniques to efficiently train models from geo-distributed datasets.

Distributed System	**Context**	**ML Role**	**Modeling Technique**
CURATOR	Distributed Storage System	Policy	Reinforcement Learning
ADARES	Virtual Execution Environment	Mechanism	Contextual Bandits
PULPO	Geo-Distributed Machine Learning	First-class Citizen	ML-System Co-Design

Table 1.1: Contributions Summary

1.6.1 ML-based Policies: CURATOR

Today's cluster storage systems embody significant functionality in order to support the needs of enterprise clusters. For example, they provide automatic replication and recovery to deal with faults, they support scaling, and provide seamless integration of both solid-state and hard disk drives. Further, they support storage workloads suited for VMs, through mechanisms such as snapshotting and automatic reclamation of unnecessary data, as well as perform space-saving transformations such as dedupe, compression, erasure coding, etc.

Closer examination of these tasks reveals that much of their functionality can be performed in the background. Based on this observation, herein, we present the design and implementation of CURATOR, a background self-managing layer for storage in enterprise clusters. In particular, we perform this work in the context of a commercial enterprise cluster product developed by Nutanix. Nutanix is a provider of enterprise clusters, which blends web-scale engineering and consumer-grade design to natively converge server, storage, virtualization, and networking, into a resilient, software-defined solution.[1]

CURATOR is an extensible, flexible, and scalable framework that was developed over a period of five years, and has been deployed on thousands of enterprise clusters. CURATOR is mainly comprised of two core components:

1. A background execution framework for cluster management tasks, where all the tasks can be expressed as MapReduce-style operations over the corresponding data.

2. A replicated and consistent key-value store, where all the important metadata of the storage system is maintained.

Herein, we report on the performance of the system. We find that the system performs garbage collection and replication effectively, balances disks, and makes storage access efficient through a number of optimizations. The resulting framework is general enough to incorporate a wide variety of background transformations.

[1]For more details refer to http://www.nutanix.com.

Nonetheless, we notice that the deployed scheduling heuristics do not necessarily work well in all clusters as there is significant heterogeneity across them as well as important workload fluctuations over time. Therefore, we propose to augment CURATOR with an *ML-based policy* to address the issues of when should these background management tasks be performed and how much work they should do. Our ML-based scheduling policy uses reinforcement learning to drive the scheduling decisions, leverages historical data from real clusters to speed up training, and has the ability to learn over time and adapt to (changing) workload characteristics.

We focus our efforts on the following tiering question: how much data to keep in SSDs and HDDs? Empirical evaluation on five simulated workloads on real clusters confirms the general validity of our approach, and shows up to $\sim$35% improvements in SSD hits and up to $\sim$20% latency reductions over threshold-based approaches.

1.6.2 ML-based Mechanisms: ADARES

Virtual execution environments are widely used in industry as they provide a high degree of flexibility and allow efficient use of cluster resources. An application that might otherwise require a dedicated server to run, can be deployed as a virtual machine (VM) and executed together with other VMs on the same physical hardware, thus enabling more efficient use of resources [210].

There are however many hurdles in achieving both high system efficiency and optimal VM performance. For example, users typically allocate resources to VMs based on guesswork, which hardly matches the actual resource needs of the applications. Even more, the application workload for a VM typically changes over time [24, 76, 64, 102], rendering static resource allocation settings inappropriate.

Incorrect resource allocations can result in a variety of problems. VMs that are not provided enough resources could experience significant application level penalties, such as trashing or swapping. Further, VMs that underutilize their resources could affect the overall system efficiency, whereas VMs that starve resources could potentially damage other VMs, which could have otherwise benefited from those extra resources [215, 216, 23, 214]. This motivates the need for a system that adaptively changes the amount of system resources allocated to each VM in a cluster.

Herein, we first perform a large-scale measurement study of clusters to characterize the resource needs for VMs in the real-world. We gather an extensive dataset by instrumenting more than 3.6K enterprise clusters running the same commercial computation and storage virtualization product developed by Nutanix as before. Our analysis allows us to quantify the extent to which user-configured resource allocations are incorrect and the overall impact on cluster efficiency. Among our main findings, we observe VM instances with significant amounts of overprovisioning as well as some underprovisioning. Further, we find significant variation across time and VMs within a cluster, which renders static resource allocations ineffective.

Unlike most existing traces [54, 172, 224, 149], our data refers to privately managed, enterprise clusters that are provisioned and operated independently by 2k+ different companies. Such environments have received little attention despite representing an important virtualization environment that is extensively used by companies [174]. Furthermore, the traces we collect contain a richer set of metrics (e.g., VM memory usage, effective I/O operations, etc.) than most other traces, enabling a more thorough analysis of the resource allocation problem.

Based on our findings, we design and build ADARES, an adaptive system that leverages an *ML-based mechanism* to automatically optimize VM resource allocations in real clusters. ADARES uses the multi-armed bandit framework with contextual information [125], also known as contextual bandits, to dynamically tune VM resources, namely virtual CPUs (vCPUs) and memory. By design, the contextual bandits framework allows a cluster manager to adapt to the VM workload characteristics through online learning, and represents a natural half-way point between supervised learning and reinforcement learning [6, 131, 28, 125].

A key challenge in leveraging contextual bandits in our setting is the "unsafe" exploration that is required for learning something useful. In other words, we need to be careful of the changes we perform to the VMs as we do not want to (permanently) impair them. To address this challenge, we build a cluster simulator from data collected by running different benchmarks in experimental clusters. We then initialize (or warm-up) our model(s) offline using the simulator, and *transfer* the knowledge gained in the simulated environment to the real clusters, in order to conduct safer configuration changes as well as speeding up training [161, 85], which translates into up to 2$\times$

resource savings when compared to models learned from scratch. We also leverage the cluster's instrumentation by providing our model a full picture of the cluster, node and VM states, so that it can make more informed decisions.

Empirical evaluation on synthetic workloads on real clusters shows that our ML-based mechanism reduces compute and memory allocations by up to 35% and 60% respectively, while achieving more predictable VM-level performance, when compared to other non-trivial baselines.

1.6.3 ML-System Co-Design: PULPO

Modern organizations have a planetary footprint. Data is created where users and systems are located, all around the globe. The reason for this is mainly two-fold: 1) minimizing latency between serving infrastructure and end-users, and 2) respecting regulatory constraints, that might require data about citizens of a nation to reside within the nation's borders.

On the other hand, many machine learning applications require access to all that data at once to build accurate models. For example, fraud prevention systems benefit tremendously from the global picture in both finance and communication networks, recommendation systems rely on the maximum breadth of data to overcome cold start problems, and the predictive maintenance revolution is only possible because of data from all markets. These types of applications that deal with geo-distributed datasets belong to a class of learning problems, which we call geo-distributed machine learning (GDML).

The state-of-the-art approach machine learning from decentralized datasets is to centralize them. This involves a two-step process: 1) the various partitions of data are copied into a single data center—thus recreating the overall dataset in a central location, and 2) learning takes place there, using existing intra-data center technologies. Based on conversations with practitioners at Microsoft, we gather that this *centralized* approach is predominant in most practical settings. This is consistent with reports on the infrastructures of other large organizations, such as Facebook [206], Twitter [130], and LinkedIn [17].

The reason for its popularity is two-fold, on the one hand, centralizing the data is the easiest way to reuse existing machine learning frameworks [231, 137, 132]), and on the other hand, machine learning algorithms are notoriously communication-intensive, and thus assumed to be non-amenable to cross-data center execution.

The centralized approach has two key shortcomings:

1. It consumes large amounts of cross-data center (X-DC) bandwidth (in order to copy the raw data to a single location). Wide-area network bandwidth has been shown to be scarce, expensive, and growing at a slower pace than most other intra-data center (in-DC) resources [169, 217, 126, 95].

2. It requires raw data to be copied across data centers, thus potentially across national borders. While international regulations are quickly evolving, we speculate that the growing concerns regarding privacy and data sovereignty [87, 67, 176, 75] might become a key limiting factor to the applicability of centralized learning approaches.

We hypothesize that both challenges will persist or grow in the future [219, 108].

In this work, we propose PULPO, a system that enables geo-distributed learning, where raw data is kept in place, and learning tasks are executed in a cross-data center fashion. We show that, by *co-designing the system with machine learning*, PULPO can achieve orders of magnitude lower cross-data center bandwidth consumption in many practical settings. In particular, PULPO leverages *optimization-based techniques* [139] together with a distributed resource management fabric to perform efficient geo-distributed training.

Moreover, as the geo-distributed learning approach PULPO enables does not require to copy raw data outside their native data center (only statistics and estimates are copied), it is structurally better positioned to deal with evolving regulatory constraints. A detailed study of this legal aspect of the geo-distributed approach is beyond the scope of this work.

The solution we propose serves as an example for a new generation of co-designed systems and learning algorithms, and allows us to present the first study on the relative efficiency of centralized versus geo-distributed approaches. In this work, we concentrate on two key metrics: cross-data center bandwidth consumption and learning runtime. Note that while the above metrics are of great practical relevance, many other dimensions (e.g., resilience to catastrophic data center failures) are worth considering when comparing alternative approaches.

We show experimentally that properly designed centralized solutions can achieve faster learning times (when the data copy latency is hidden by streaming data as it becomes available), but that co-designed distributed solutions with ML can achieve much lower cross-data center bandwidth utilization, and thus substantially lower cost for large-scale learning.

1.7 Organization

The remainder of this thesis is structured as follows:

Chapter 2 introduces ML background information and related work of each of the techniques used.
Chapter 3 introduces CURATOR and describes its reinforcement-learning task scheduling policy.
Chapter 4 describes ADARES and its bandit-based mechanism to adjust VM resources.
Chapter 5 presents PULPO, and its co-design with ML to reduce communication costs.
Chapter 6 concludes.

2 | Background and Related Work

In this chapter we present the necessary background information and general related work of systems optimized using machine learning. The literature on methods for machine learning can be overwhelming, and different techniques are more appropriate for different scenarios. Broadly speaking, we can encompass machine learning into three main types: supervised learning, unsupervised learning, and reinforcement learning.

Besides the above three, another very powerful framework, though oftentimes overlooked in the systems community, is the bandit framework. In particular, we are interested in contextual bandits, which can be considered a natural half-way point between supervised learning and reinforcement learning. The construction of context using features comes from supervised learning, while exploration, necessary for good performance, is inherited from reinforcement learning [6]. Herein, we provide an overview of supervised learning, contextual bandits, and reinforcement learning.

2.1 Supervised Learning-based Techniques

2.1.1 Background

Although we do not use supervised learning per se to optimize any system in this work, we do leverage optimization-based techniques that can be used for training supervised learning models. Thus, we include a brief overview of supervised machine learning here as well as an introduction to distributed methods.

In supervised learning, the goal is to learn a mapping from inputs x to outputs y, given a labeled dataset $D = \{x_i, y_i\}_{i=1}^{N}$, where N is the number of training examples. In general, each training input x_i is a d-dimensional vector of numbers, called feature vector, and each output y_i is either a categorical or nominal variable (e.g., spam or ham), in which case is called a classification problem, or a real-valued scalar (e.g., salary), in which case is called a regression problem [155].

The learning goal can be expressed as an objective function (e.g., low error when predicting a person's salary). Generally, by minimizing this function, the ML algorithm obtains a model (e.g., a d-dimensional vector of numbers, where each of the numbers is called a model parameter) capable of making predictions on future unseen inputs. In general, there is no closed-form solution, thus the algorithm iteratively refines the model using some mathematical optimization technique in order to approach the optimal solution [132].

However, oftentimes, training a model in a single machine might become infeasible, mainly in terms of training time, provided the dataset is large enough [127]. In such context, distributed machine learning (DML) emerged as a natural way to scale out learning algorithms [163], where the dataset is partitioned among multiple worker nodes. Note that the vast majority of datasets are horizontally partitioned, wherein subsets of examples are stored at different nodes. In general, learning proceeds in communication rounds. At each round, a server node sends the current algorithm state (e.g., the model parameters) to the workers. Each worker node then performs some local computation based on the state received and its shard of the dataset, and sends an update to the server. The server applies the updates to the algorithm state, and the process repeats [144].

Traditional DML typically focuses on distributing computation across machines within a single data center to accelerate training, and many systems offer different abstractions for running learning algorithms in this mode [132, 137, 146, 4, 225, 49].

2.1.2 Related Work

A considerable amount of literature has been published on applying different supervised learning techniques into systems.

There has been some work on failure and troubleshooting of systems using supervised learning. For example, the work by Fulp et al. [83] focuses on predicting computer system failures using support vector machines (SVMs) [40]. CLUEBOX [179] leverages available performance logs to characterize workloads, predict performance, and discover anomalous behavior using random forests [36]. Other studies apply ML techniques to make systems more secure. One such system is HeatRay [68], which uses SVMs for driving security configuration changes to combat identity snowball attacks.

More recent applications of ML into systems encompass areas such as data structure design [119], microarchitecture [97], compilers [57], and caches [26]. Kraska et al. [119] uses linear models and neural networks to enhance/replace indexed structures, such as B-Trees or Bloom filters, by leveraging the distribution of the data being indexed. Hashemi et al. [97] treats prefetching as a machine learning classification problem. They use deep learning to predict future memory accesses that will miss in the on-chip cache and access memory based on past history.

The work by Bortnikov [32] proposes using gradient-boosted decision trees [82] to predict the slowdown of a task compared to other similar tasks in order to proactively avoid stragglers in distributed analytical systems. Within the same line of work, Yadwadkar et al. [226, 227] uses SVMs to predict if a node is too busy to finish a task in a timely manner. They then modify the cluster scheduler to use those predictions so as to avoid creating stragglers.

Further, DeepMind created an efficient and adaptive framework to understand data centre dynamics and optimize efficiency using ML [60]. In particular, they leverage historical data such as temperatures, power, pump speeds, to train an ensemble of neural networks to improve power usage effectiveness.

2.2 Bandit-based Techniques

2.2.1 Background

2.2.1.1 Contextual Bandits

There is another type of learning problem that falls under the category of multi-armed bandit (MAB) with contextual information. In this problem, also known as contextual bandits, an agent collects rewards for actions taken over a sequence of rounds. In each round, the agent chooses the action to take based on: 1) *context* (or features) of the current round, and 2) *feedback* (or rewards) obtained in the previous rounds. In any given round, the agent observes *only* the reward for the chosen action, thus the feedback is said to be *partial* [6].

More formally, the learning agent proceeds in a sequence of discrete trials, $t = 1,2,3...$ At each trial t, the agent observes the context x_t, and selects an action, $a_t \in \mathscr{A}_t$, where $\mathscr{A}_t$ is the set of all actions available at time t. The agent then receives a reward, $r_{t,a_t} \in [0,1]$, and improves its action-selection strategy with the tuple (x_t, a_t, r_{t,a_t}) [131].

The total reward for the agent after T trials is defined as $\sum_{t=1}^{T} r_{t,a_t}$. Similarly, the optimal expected T-trial reward is defined as $\mathbf{E}[\sum_{t=1}^{T} r_{t,a_t^*}]$, where a_t^* is the action with the maximum expected reward at trial t. The goal of the agent is to maximize the expected reward, or, equivalently, minimize the *regret* with respect to the optimal action-selection strategy. The regret of the agent after T trials is formally defined as follows:

$$R(T) = \mathbf{E}[\sum_{t=1}^{T} r_{t,a_t^*}] - \mathbf{E}[\sum_{t=1}^{T} r_{t,a_t}] \tag{2.1}$$

A fundamental challenge in bandit problems is the need for balancing exploration and exploitation. In order to minimize the regret in Equation 2.1, the agent *exploits* its past experience and chooses the action that appears to be the best. However, that action might be sub-optimal due to the agent's insufficient knowledge. Instead, the agent may need to *explore* by selecting seemingly sub-optimal actions in order to gather more knowledge about them [131].

Common applications of contextual bandits include, but are not limited to, personalized news recommendations, clinical trials, and mobile health interventions [203, 28]. There are many algorithms for contextual bandits, such as EXP4 [38], Epoch-Greedy [125], and LinUCB [131]. We describe the latter next.

2.2.1.2 LinUCB

LinUCB [131] is an upper confidence bound (UBC) algorithm. In trial t, these algorithms [7, 16] estimate both the mean reward $\hat{\mu}_{t,a}$ of each action a as well as a confidence interval $c_{t,a}$, so that $|\hat{\mu}_{t,a} - \mu_a| < c_{t,a}$ holds with high probability. Then, they select the action with the highest upper confidence bound $a_t = \arg\max_a(\hat{\mu}_{t,a} + c_{t,a})$.

Given some parametric form of reward function, different methods exist to estimate the confidence interval of the parameters from which we can compute the UCB of the different actions. However, those approaches are typically expensive as they discard the structure of the contexts, even though the contexts often have structure [38].

Li et al. [131] shows that a confidence interval can be computed efficiently in closed form when the payoff model is linear. LinUCB assumes that the reward given a context follows a linear structure with noise. Although this is a strong assumption, it makes the problem computationally tractable and works well in practice.

2.2.2 Related Work

In recent years, there has been an increasing amount of literature on using bandit-based techniques to optimize systems. In particular, bandits have gained more attraction in the database community. For example, Cuttlefish [109] supports adaptive processing of online database query plans using the multi-armed bandit framework. The system explores candidate physical operator instances during query execution and exploits the fastest ones using Thompson sampling [143, 8]. Another adaptive query processing framework that leverages bandits is Micro Adaptivity [170], conceived as part of Vectorwise [237]. It uses epoch-based ε-greedy bandit policies to select between several black-box “flavors” of vectorized operators at each function call. OtterTune [209] is an autonomic DBMS that leverages Gaussian Processes [194] to effectively explore/exploit the high-dimensional space of DBMS configuration parameters.

Pytheas [107] leverages bandits to optimize quality of experience in networked applications, such as video streaming, internet telephony, and social networks. RE^{X} [164] is a framework for self-adaptive software architectures that uses Thompson sampling to solve the search space explosion problem inherent in runtime emergent software. By using a bandit-based approach, the authors show how a web server can be autonomously assembled from discovered parts, and how the system can subsequently be optimized by seamlessly reassembling it from alternative components.

SiblingRivalry [13] applies bandit-based techniques to allow parallel programs to continuously adapt and optimize themselves to their execution environment. Similarly, OpenTuner [12] also uses bandits for program auto-tuning to help narrowing down the search space; that is, techniques which perform well by finding better configurations are allocated larger budgets of tests to run, whereas techniques which perform poorly are allocated fewer tests or disabled entirely.

JouleGuard [100] is a runtime control system that coordinates approximate applications with system resource usage to optimize energy consumption. It identifies the most energy efficient system configuration using the multi-armed bandit framework.

Recent work leverages contextual bandits to perform off-policy evaluation in distributed systems, i.e., it uses already-collected data from deployed policies in current systems to evaluate new candidate policies offline [129]. In their work, the authors develop a methodology for harvesting existing randomness without intervening in live systems. In particular, they apply their methodology to machine health monitoring in Microsoft Azure, load balancing, and caching problems.

Finally, the popular system NEXT [106] facilitates the development, testing, and deployment of bandit algorithms (and active learning in general) for real applications.

2.3 Reinforcement Learning-based Techniques

2.3.1 Background

2.3.1.1 Reinforcement Learning

Reinforcement learning (RL) can be thought of an extension of contextual bandits, where the actions change the state of the world. More formally, the agent interacts with its environment in a sequence of discrete time steps, $t = 0, 1, 2, 3....$ At each time step t, the agent senses the environment's state, $s_t \in S$, where S is the set of all possible states, and selects an action, $a_t \in \mathscr{A}_{s_t}$, where $\mathscr{A}_{s_t}$ is the set of all actions available in state s_t. The agent receives a reward, $r_{t+1} \in \mathbb{R}$, and finds itself in a new state, $s_{t+1} \in S$.

RL is about learning a policy π that maps situations to actions, so as to maximize a numerical reward signal over the long run. If the sequence of rewards received after time step t is $r_{t+1}, r_{t+2}, r_{t+3}, ...$, then the objective of learning is to maximize the *expected discounted return.*

The discounted return R_t is given by:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

where $0 \leq \gamma \leq 1$ is called the discount factor. $\gamma = 0$ will make the agent "myopic" (or short-sighted) by only considering immediate rewards, while $\gamma \to 1$ will make it strive for a long-term high reward [199].

At every time step t, the agent is not told which actions to take, but instead, it must discover which actions yield the most reward by trying them out [199]. As with contextual bandits, there are two things that are useful for the agent to do, known as the exploration/exploitation trade-off. It can either *exploit* the knowledge that it has and find a good policy with respect to this knowledge, at the risk of missing some large reward out there, or it can *explore* the state space in search of a region with more reward, at the risk of wasting time or collecting punishments.

In general, reinforcement learning can be applicable to situations where we cannot immediately reject the hypothesis that, when taking action a_t in state s_t, the next state s_{t+1} and reward r_{t+1} is independent and identically distributed (iid), and this can be determined with statistical tests.

Many RL algorithms involve estimating *value functions*, which are functions of states (or state-action pairs) that estimate how good, in terms of expected discounted return, it is for the agent to be in a given state (or how good it is to perform a given action in a given state). As the rewards an agent will expect in the future depend on the actions it will take, *value functions* are defined with respect to policies (i.e., the mapping of states to actions). Roughly speaking, RL is about finding the policy π^* that achieves the highest reward over the long run, known as the optimal policy [199].

There are many RL algorithms, such as policy gradient methods [200], dynamic programming methods [27], and Q-learning [220]. We describe the latter next.

2.3.1.2 Q-learning

Q-learning [220] is a model-free reinforcement learning algorithm, which falls under the class of temporal difference methods [201, 198], where an agent tries an action a_t at a particular state s_t, and evaluates its effects in terms of the immediate reward r_{t+1} it receives and its estimate of the value of the state s_{t+1} to which it is taken. By repeatedly trying all actions in all states, it learns which ones are best, i.e., it learns the optimal policy π^*, judged by long-term discounted return.

Q-learning uses a function Q that accepts a state s_t and action a_t, and outputs the value of that state-action pair, which is the estimate of the expected value (discounted return) of doing action a_t in state s_t and then following the optimal policy π^* [122]. Its simplest form, one-step Q-learning, is given by:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.3}$$

or, equivalently:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_{t+1} + \gamma max_a Q(s_{t+1}, a)) \tag{2.4}$$

where $0 \leq \alpha \leq 1$ is the learning rate, and determines to what extent the new information will override the old one. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the newly acquired information.

The Q-function above can be implemented using a simple lookup table. Nevertheless, when the state-action space is large, e.g., continuous spaces, storing Q-values in a table becomes intractable. The Q-function needs to be approximated by a function approximator. The compression achieved by a function approximator allows the agent to generalize from states it has visited to states it has not. The most important aspect of function approximation is not just related to the space saved, but rather to the fact that it enables generalization over input spaces [178].

Q-learning tends to work well when the state space is large, can be naturally implemented in an online fashion, and has the ability to learn without requiring a model of the environment [11, 199].

2.3.2 Related Work

Recent successes in deep reinforcement learning [153, 152] have made it increasingly popular among the systems community. Applying RL to optimize systems covers a wide spectrum of systems, in particular, deep reinforcement learning has been shown to vastly outperform traditional heuristics in topics such as scheduling [148, 147], resource management [141], and data center traffic optimization [47].

The work by Mirhoseini et al. [148] introduces a hierarchical model for placement of computational graphs onto different hardware devices, such as CPUs and GPUs. They use deep reinforcement learning to learn graph operation assignments to groups and to allocate those groups to available devices. Their agent is optimized for speed of computation and for feasibility, i.e., to have sufficient memory available on each device for the computation assigned.

DeepRM [141] uses RL to train a neural network for multi-dimensional resource packing. In particular, DeepRM uses policy gradient methods [200] to optimize various objectives such as minimizing average job slowdown or completion time, and performs comparably to state-of-the-art heuristics while being able to adapt to different conditions.

In AuTO [47], the authors propose using deep reinforcement learning for traffic optimizations in data centers, specifically, flow scheduling and load balancing. Their system can adapt to voluminous, uncertain, and volatile data center traffic, while achieving operator-defined goals. Further, the work by Boyan et al. [34] proposes Q-routing, an algorithm for packet routing, in which a reinforcement learning module is embedded into each node of a switching network, where routing decisions lead to minimal delivery times when compared to non-adaptive algorithms based on precomputed shortest paths.

Reinforcement learning-based techniques have also been used for adaptive video streaming. In particular, Pensieve [142] trains a neural network using A3C [151], a state-of-the-art actor-critic RL algorithm, to select bitrates for future video chunks based on observations collected by client video players. Li et al. [134] proposes an end-to-end cooling control system also based on the actor critic framework and an offline version of the deep deterministic policy gradient algorithm [135] to minimize cooling consumption in data centers; an alternative approach to Google's DeepMind [60].

3 | CURATOR

Modern cluster storage systems perform a variety of background tasks to improve the performance, availability, durability, and cost-efficiency of stored data. For example, cleaners compact fragmented data to generate long sequential runs, tiering services automatically migrate data between solid-state and hard disk drives based on usage, recovery mechanisms replicate data to improve availability and durability in the face of failures, cost saving techniques perform data transformations to reduce the storage costs, and so on.

In this work, we present CURATOR, a background MapReduce-style execution framework for cluster management tasks, in the context of a distributed storage system used in Nutanix enterprise clusters. We describe CURATOR's design and implementation, and evaluate its performance using a handful of relevant metrics.

Finally, we propose augmenting CURATOR's scheduler with an *ML-based policy* to decide when to execute the management tasks, which can adapt to varying workload characteristics. In particular, we use *reinforcement learning* to identify efficient execution policies. Our approach leverages historical data from real clusters to speed up training, and shows performance improvements over other baselines. For example, empirical evaluation on simulated workloads on real clusters shows latency reductions of up to $\sim$20% when compared to a threshold-based approach.

In summary, the main contributions of this work are:

- We provide an extensive description of the design and implementation of CURATOR, an advanced distributed cluster background management system, which performs, among others, data migration between storage tiers based on usage, data replication, disk balancing, garbage collection, etc.

- We present measurements on the benefits of CURATOR using a number of relevant metrics, e.g., latency, I/O operations per second (IOPS), disk usage, etc., in a contained local environment as well as in customer deployments and internal corporate clusters.

- Finally, we propose an ML-based policy to augment CURATOR's scheduler. Our approach uses reinforcement learning to decide when to trigger the background maintenance tasks and leverages already-collected data from production clusters to accelerate training. Empirical results on a storage tiering task demonstrate the benefits of our solution.

We structure this chapter as follows: Section 3.1 presents an overview of the cluster architecture and introduces the data structures used to store the data and metadata in the storage system. Then, we provide an extensive description of the design and implementation of CURATOR, together with our reinforcement learning formulation in Section 3.2. We show evaluation results in Section 3.3 related work in Section 3.4, and summarize in Section 3.5.

3.1 Distributed Storage for Enterprise Clusters

We perform this work in the context of a distributed storage system designed by Nutanix for enterprise clusters. In this section, we provide an overview of the software architecture, the key features provided by the storage system, and the data structures used to support them. Herein, we present the necessary background information for understanding the design of CURATOR.

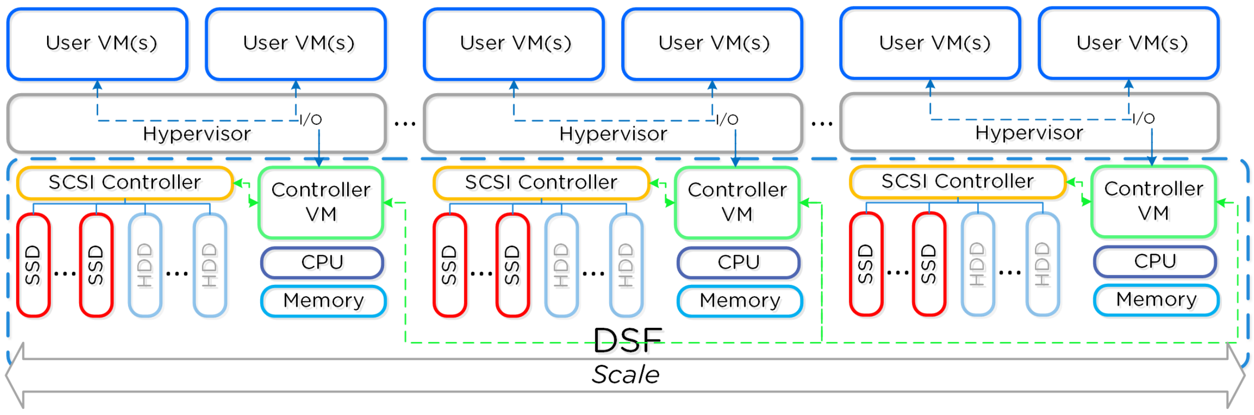


Figure 3.1: Cluster Architecture and Distributed Storage Fabric. UVMs access the storage devices distributed across the cluster using CVMs [204].

3.1.1 Clusters Architecture

Nutanix is a well-known provider of enterprise cloud platforms. Their software architecture is designed for clusters of varying sizes. They have cluster deployments at a few thousand different customer locations, with cluster sizes typically ranging from a few nodes to a few dozens of nodes. Cluster nodes might have heterogeneous resources, since customers add nodes based on need.

Their clusters support virtualized execution of (legacy) applications, typically packaged as VMs. The cluster management software provides a management layer for users to create, start, stop, and destroy VMs. Further, this software automatically schedules and migrates VMs taking into account the current cluster membership and the load on each of the individual nodes. These tasks are performed by a controller virtual machine (CVM) running on each node in the cluster.

The CVMs work together to form a distributed system that manages all the storage resources in the cluster. The CVMs and the storage resources that they manage provide the abstraction of a distributed storage fabric (DSF) that scales with the number of nodes and provides transparent storage access to user VMs (UVMs) running on any node in the cluster.[1]

Figure 3.1 shows a high-level overview of the cluster architecture. Applications running in

[1]We use VMs and UVMs interchangeably throughout this thesis.

VMs access the DSF using legacy filesystem interfaces (such as NFS, iSCSI, or SMB). Operations on these legacy filesystem interfaces are interposed at the hypervisor layer and redirected to the CVM. The CVM exports one or more block devices that appear as disks to the VMs. These block devices are virtual (they are implemented by the software running inside the CVMs), and are known as vDisks. Thus, to the VMs, the CVMs appear to be exporting a storage area network (SAN) that contains some disks on which the operations are performed.

Unlike SAN/NAS and other related solutions (e.g., OneFS [74], zFS [175], GlusterFS [89], LustreFS [196], GPFS [181]), the cluster nodes serve as both VM compute nodes as well as storage nodes. All user data (including the operating system) in the user VMs resides on these vDisks, and the vDisk operations are eventually mapped to some physical storage device (SSDs or HDDs) located anywhere inside the cluster.

Crucially, the cluster management software has a comprehensive view regarding cluster state and thus can be instrumented to provide valuable measurement data. We can collect data regarding resource utilization on different nodes and VMs (e.g., CPU, memory, storage), the number of VMs running on a node, the I/O operations performed (since all data access is mediated by the cluster storage layer), as well as cluster health attributes. We use this data in our study, collected from different layers of the Nutanix cluster architecture.

3.1.2 Storage System and Associated Data Structures

We now describe the key requirements of the distributed storage fabric and how these requirements influence the data structures used for storing the metadata and the design of CURATOR.

R1 *Reliability/Resiliency:* the system should be able to handle failures in a timely manner.

R2 *Locality preserving:* data should be migrated to the node running the VM that frequently accesses it.

R3 *Tiered Storage:* data should be tiered across SSDs, hard drives, and the public cloud. Further, the SSD tier should not merely serve as a caching layer for hot data, but also as permanent storage for user data.

R4 *Snapshot-able:* the system should allow users to quickly create snapshots for greater robustness.

R5 *Space efficient:* the system should achieve high storage efficiency while supporting legacy applications and without making any assumptions regarding file sizes or other workload patterns.

R6 *Scalability:* the throughput of the system should scale with the number of nodes in the system.

The above set of requirements manifest in the design of CURATOR in two ways: 1) the set of data structures that are used for storing the metadata, and 2) the set of management tasks that will be performed by the system. We discuss the data structures below and defer the management tasks performed by CURATOR to Section 3.2.3.

Each vDisk introduced in Section 3.1.1 corresponds to a virtual address space forming the individual bytes exposed as a disk to user VMs. Thus, if the vDisk is of size 1 TB, the corresponding address space maintained is 1 TB. This address space is broken up into equal sized units called vDisk blocks. The data in each vDisk block is physically stored on disk in units called extents. Extents are written/read/modified on a sub-extent basis (also known as slice) for granularity and efficiency. The extent size corresponds to the amount of live data inside a vDisk block; if the vDisk block contains unwritten regions, the extent size is smaller than the block size (thus satisfying R5).

Several extents are grouped together into a unit called an extent group. Each extent and extent group is assigned a unique identifier, referred to as extentID and extentGroupID respectively. An extent group is the unit of physical allocation and is stored as a file on disks, with hot extent groups stored in SSDs and cold extent groups on hard drives (R3). Extents and extent groups are dynamically distributed across nodes for fault-tolerance, disk balancing, and performance purposes (R1, R6).

Given the above core constructs (vDisks, extents, and extent groups), we now describe how the system stores the metadata that helps locate the actual content of each vDisk block. The metadata maintained by the storage system consists of the following three main maps:

- *vDiskBlock map:* maps a vDisk and an offset (to identify the vDisk block) to an extentID. It is a logical map.

- *extentID map:* maps an extent to the extent group that it is contained in. This is also a logical map.

- *extentGroupID map:* maps an extentGroupID to the physical location of the replicas of that extentGroupID and their current state. It is a physical map.

Here are a few implications regarding the aforementioned data structures. Multiple vDisks created through snapshots can share the same extent. The vDiskBlock map of a snapshot can either directly point to an extent shared with a prior snapshot or have a missing entry, in which case the vDiskBlock map of the previous snapshot is consulted. This facility allows for instantaneous creation of snapshots, i.e., the system can create an empty vDiskBlock map entry and have it point to the previous snapshot for all of its unfilled entries (R4). At the same time, it enables a later optimization of metadata lookup using lazy filling of the missing entries. When a vDisk block is updated on the new snapshot, a new extent is created to hold the updated data.

The level of indirection introduced by the extentID map allows efficient updates whenever data from one extent group is relocated to another (e.g., to optimize access), as it is a single place in which we store the physical extentGroupID in which the extent is located (thus aiding R2, R3).

Finally, a set of management operations can be performed by only consulting the extent-GroupID map. For example, the system can detect (and repair) if the number of replicas for a given extentGroupID falls under certain threshold by only accessing this map (the logical maps will remain untouched), thus addressing R1.

Overall, the resulting data structures set up CURATOR to perform various management tasks in an efficient and responsive manner.

3.2 System Design

This section describes the design of CURATOR, a system that safeguards the DSF's health and performance by executing background maintenance tasks. This section starts with a high-level description of the goals that determined the design of CURATOR, and then provide details of its architecture. We then present the tasks it performs, the policies under which those tasks are executed, and demonstrate CURATOR's value with a set of empirical results of clusters in the wild. Finally, we propose a reinforcement learning-based formulation as a new scheduling policy for triggering the maintenance tasks.

3.2.1 Goals

CURATOR is the cluster management component responsible for managing and distributing various storage management tasks throughout the cluster, including continuous consistency checking, fault recovery, data migration, space reclamation, and many others. CURATOR oversees the overall state of cluster storage and takes actions as necessary.

CURATOR's design is influenced by the following considerations:

- *Scalable:* The system should scale with the amount of storage served by the storage system and cope with heterogeneity in node resources.

- *Flexible and generic:* The system should provide a flexible and extensible framework that can support a broad class of background maintenance tasks.

- *Non-interfering:* CURATOR's mechanisms should not interfere with nor complicate the operations of the underlying storage fabric.

3.2.2 Components

Based on the above considerations, CURATOR's design encompasses four key components and/or concepts.

3.2.2.1 Distributed Metadata

The metadata (i.e., the maps introduced in Section 3.1.2) is stored in a distributed ring-like manner, based on a heavily modified Apache Cassandra [123], enhanced to provide strong consistency for updates to replicated keys. Paxos [124] is utilized in order to guarantee correctness.

The decision behind having the metadata distributed lies in the fact that the system should not be bottlenecked by metadata operations. Although a distributed key-value store requires more hard work from the perspective of processing the metadata, it provides a way to scale from small clusters (say three nodes) to larger (hundreds of nodes) ones. However, many other commercial storage products have decided to keep the metadata in a single node; we observe two main issues with this approach: 1) special dedicated nodes for metadata cause a single point of failure, and 2) the vertical scale up requirement of such nodes—as the physical size of these storage nodes increases with the number of logical entities, they will need to be replaced or upgraded in terms of memory/CPU.

3.2.2.2 Distributed MapReduce Execution Framework

CURATOR runs as a background process on every node in the cluster using a master/slave architecture. The master is elected using Paxos, and is responsible for task and job delegation. CURATOR provides a MapReduce-style infrastructure [59] to perform the metadata scans, with the master CURATOR process managing the execution of MapReduce operations. This ensures that CURATOR can scale with the amount of cluster storage, adapt to variability in resource availability across cluster nodes, and perform efficient scans/joins on metadata tables. Note that any metadata stored in a distributed key-value store should be able to utilize this MapReduce framework.

Although this framework bears resemblance to some data-parallel engines, such as Hadoop or Spark, the reason behind writing it from scratch instead of re-purposing an existing one was two-fold: 1) efficiency, as most of these open-source big data engines are not fully optimized to make a single node or a small cluster work efficiently, instead, they assume they will have enough compute as their deployments tend to scale out, and 2) their requirement of a distributed storage system (e.g., HDFS), a recursive dependence that Nutanix did not want to have in the clustered storage system.

It is worth pointing out that having a background MapReduce process to do post-process/lazy storage optimization allows to achieve better latencies for user I/O. While serving an I/O request, the DSF does not have to make globally optimal decisions on where to put a piece of data nor what transformations (compression, dedupe, etc.) to apply on that data. Instead, it could make decisions based on minimal local context, which allows to serve user I/O faster. Later on, CURATOR in the background would re-examine those decisions and make a globally optimal choice for data placement and transformation.

3.2.2.3 Co-design of CURATOR *with Underlying Storage System*

The DSF provides an extended API for CURATOR, including but not limited to the following low-level operations: migrate an extent from one extent group to another, fix an extent group so that it meets the durability and consistency requirements, copy a block map from one vDisk to another, and perform a data transformation on an extent group.

CURATOR only performs operations on metadata, and gives *hints* to an I/O manager service in the storage system to act on the actual data. It is up to the storage system to follow CURATOR's advice, e.g., it may disregard a suggestion of executing a task due to heavy load or if a concurrent storage system operation has rendered the operation unnecessary. CURATOR makes sure that the I/O manager knows the version of metadata it based its decision on. The I/O manager checks the validity of the operations based on metadata timestamps (for strong consistency tasks like garbage collection) or last modified time (for approximate tasks such as tiering). This approach also eliminates the need for CURATOR to hold locks on metadata in order to synchronize with the foreground tasks; concurrent changes only result in unnecessary operations and does not affect correctness.

3.2.2.4 Task Execution Modes and Priorities

During a MapReduce-based scan, the mappers and reducers are responsible for scanning the metadata in Cassandra, generating intermediate tables, and creating synchronous and asynchronous tasks to be performed by the DSF. Synchronous tasks are created for fast operations (e.g., delete a vDisk entry in the vDiskBlock metadata map) and are tied to the lifetime of the MapReduce job. Conversely, asynchronous tasks are meant for heavy operations (e.g., dedupe, compression, and replication) and are sent to the master periodically, which batches them, and sends them to the underlying storage system for later execution (with throttling enabled during high load).

These tasks are not tied to the lifetime of the MapReduce job. Note that although these tasks are generated based on a cluster-wide global view using MapReduce-based scans, their execution is actually done in the individual nodes paced at a rate suitable to each node's workload. The rate depends on the CPU/disk bandwidth available at each node. In other words, the system computes what tasks need to be performed in a bulk-synchronous manner, but executes them independently (in any order) per node.

3.2.3 Management Tasks

In this section, we describe how CURATOR's components work together to perform four main categories of tasks. Table 3.1 includes a summary of the categories, tasks, and metadata maps touched by each of the tasks.

3.2.3.1 Recovery Tasks

Disk Failure/Removal and Fault Tolerance In the event of a disk or node failure, or if a user simply wants to remove/replace a disk, CURATOR receives a notification and starts a metadata scan. Such a scan finds all the extent groups that have one replica on the failed/removed/replaced node/disk and notifies the underlying storage system to fix these under-replicated extent groups to meet the replication requirement. This is handled by the storage system as a critical task triggered by a high-priority event, which then aims to reduce the time that the cluster has under-replicated

Category	**Task**	**Metadata Maps**		
		vDiskBlock	**extentID**	**extentGroupID**
Recovery	**Disk Failure/Removal**			x
	Fault Tolerance			x
Data Migration	**Tiering**			x
	Disk Balancing			x
Space Reclamation	**Garbage Collection**	x	x	x
	Data Removal	x	x	x
Data Transformation	**Compression**			x
	Erasure Coding			x
	Deduplication	x	x	x
	Snapshot Tree Reduction	x		

Table 3.1: Metadata Tables accessed by Management Tasks

data. Note that these tasks require access to just the extentGroupID map and benefit from the factoring of the metadata into separate logical and physical maps.

3.2.3.2 Data Migration Tasks

Tiering This task moves cold data from a higher storage tier to a lower tier, e.g., from SSD to HDD, or from HDD to the public cloud. CURATOR is only involved in *down* migration, not *up*, i.e., it does not migrate data from HDD to SSD, or from the public cloud to HDD. Up migration, on the other hand, is done by the DSF upon repeated access to hot data. Taken together, the actions of CURATOR and DSF aim to keep only the hottest data in the fastest storage tiers in order to reduce the overall user access latency.

This task is costly as it involves actual data movement, not just metadata modifications. CURATOR computes the "coldness" of the data during a metadata scan, and notifies the DSF to perform the actual migration of the coldest pieces. The coldness is computed based on least recently used (LRU) metrics. Failing to execute the tiering task can (and surely will) lead to performance degradation in the long run.

The cold data is identified by the modified time (mtime) and access time (atime), retrieved during a scan. Both mtime (write) and atime (read) are stored in different metadata maps. The former is located in the extentGroupID map, whereas the latter resides in a special map called extentGroupIDAccess map. This latter access map was especially created to support eventual consistency for non-critical atime data (in contrast to the extentGroupID map's strict consistency requirements) and thereby improve access performance. As a consequence of being stored in separate maps, the mtime and atime of an extent group might be located in different nodes, therefore, communication may be required to combine these two attributes.

In order to compute the "coldness" of the data, a MapReduce job is triggered to scan the aforementioned metadata maps. The *map* tasks emit the extentGroupID as key, and the mtime (or atime) as value. The *reduce* tasks perform a join-like reduce based on the extentGroupID key. The reduce tasks generate the (egid, mtime, atime) tuples for different extent groups and sort these tuples to find the cold extent groups. Finally, the coldest extent groups are sent to the DSF for the actual data migration.

Disk Balancing This task moves data within the same storage tier, from high usage disks to low usage ones. The goal is to bring the usage of disks within the same tier, e.g., the SSD tier, as close as possible to the mean usage of the tier. This task not only reduces the storage tier imbalance, but also decreases the cost of replication in the case of a node/disk failure. To minimize unnecessary balancing operations, CURATOR does not execute the balancing if the mean usage is low, even if the disk usage spread is high. In case it executes the balancing, as with tiering, it only attempts to move cold data. The MapReduce scans identify unbalanced source and target disks, together with cold data, and notifies the storage fabric to perform the actual migration of extent groups.

3.2.3.3 Space Reclamation Tasks

Garbage Collection There are many sources of garbage in the storage system, e.g., when an extent is deleted but the extent group still has multiple live extents and cannot be deleted, garbage due to wasting preallocated larger disk spaces on extent groups that became immutable and did not use all of the allocated quota, when the compression factor for an extent group changes, etc. Garbage collection increases the usable space by reclaiming garbage and reducing fragmentation. It does so in three ways:

1. *Migrate Extents:* migrate live extents to a new extent group, delete the old extent group, and then reclaim the old extent group's garbage. It is an expensive operation as it involves data reads and writes. Therefore, CURATOR performs a cost-benefit analysis per extent group and chooses for migration only the extent groups where the benefit (amount of dead space in the extent group) is greater than the cost (sum of space of live extents to be migrated).

2. *Pack Extents:* try to pack as many live extents as possible in a single extent group.

3. *Truncate Extent Groups:* reclaim space by truncating extent groups, i.e., reducing their size.

Data Removal The data structures introduced in Section 3.1.2 are updated in such a way that there cannot be dangling pointers, i.e., there cannot be a vDisk pointing to an extent that does not exist, or an extent pointing to an extent group that does not exist. However, there can be unreachable data, e.g., an extent that is not referenced by any vDisk, or an extent group that is not referenced by any extent. These could be due to the side-effects of vDisk/snapshot delete operations or a consequence of failed DSF operations.

In DSF, extent groups are created first, then extents, and finally vDisks. For removal, the process is backwards; unused vDisks are removed first, then the extents, and finally the unreferenced extent groups. This task performs the removal process in stages (possibly in successive scans), and enables the reclamation of unused space in the system. Note that only the deletion of extent groups frees up physical space.

3.2.3.4 Data Transformation Tasks

Compression and Erasure Coding CURATOR scans the metadata tables and flags an extent group as a candidate for compression/coding if the current compression of the extent group is different from the desired compression type or if the extent group is sufficiently cold.

Once CURATOR identifies the extent groups (thus extents) for compression/coding, it sends a request to the DSF, which performs the actual transformation by migrating the extents. The main input parameters of this request are the set of extents to be compressed (or migrated), and the extentGroupID into which these extents will be migrated. If the latter is not specified, then a new extent group is created. This API allows to pack extents from multiple source extent groups into a single extent group. Also, instead of always creating a new extent group to pack the extents, CURATOR can select an existing extent group and pack more extents into it. The target extent groups are also identified using MapReduce scans and sorts.

Deduplication Dedupe is a slightly different data transformation, as it involves accessing other metadata maps. During a scan, CURATOR detects duplicate data based on the number of copies that have the same precomputed fingerprint, and notifies the DSF to perform the actual deduplication.

Snapshot Tree Reduction The underlying storage system supports snapshots, which are *immutable* lightweight copies of data (similar to a simlink), and can therefore generate an instantaneous copy of a vDisk. Every time the system takes a snapshot, a new node is added to a tree, called the snapshot tree, and the vDisk metadata is inherited. Snapshot trees can become rather deep. In order to be able to read a leaf node from a tree, the system needs to traverse a sequence of vDiskBlock map entries. The bigger the depth of a tree, the more inefficient the read operation becomes. To address this, the snapshot tree reduction task "cuts" the snapshot trees, by copying vDiskBlock map metadata from parents to child nodes. There are two flavors, *partial* and *full*, and their use depends on whether we need vDisk metadata only from some ancestors (*partial*) or from all of them (*full*). Once the copy is done, the child vDisks have all the information needed for direct reads, i.e., there is no need to access the ancestors' metadata, thus, the read latency is reduced.

3.2.4 Scheduling Policies

The tasks described above are executed based on (roughly) four different policies.

Event-driven These tasks are triggered by events. For example, whenever a disk or node fails, a recovery task is executed, no matter what. These are critical, higher priority tasks.

Threshold-based These are dynamically executed tasks based on fixed thresholds violations. For example, when the tier usage is “high”, or the disk usage is “too” unbalanced, etc. We provide both examples below.

In order to be eligible for the tiering task, the storage tier usage from where CURATOR wants to down migrate the data should exceed a certain (preconfigured) threshold. Similarly, in order to be considered for balancing, the mean tier usage and the disk usage spread should both exceed certain thresholds. The disk usage spread is defined as the difference between the disk with maximum usage and the disk with minimum usage within the tier.

Periodic Partial We next consider tasks that are neither triggered nor threshold-driven, and access only a subset of the metadata maps. These tasks are executed every h_1 hours, and are grouped based on the metadata tables they scan.

Periodic Full All tasks are executed as part of a *full* scan every h_2 hours. This policy is called *full* as it scans all three metadata tables in Cassandra, the vDiskBlock, extentID, and extentGroupID maps. Because the *partial* scan only works on a subset of the metadata maps, it can run more frequently than the *full* scan, i.e., $h_1 < h_2$. In general, scans are expensive, hence, when a scan is running, CURATOR tries to identify as many asynchronous tasks as possible and lets them drain into the DSF over time. In other words, CURATOR combines the processing that must be done for the different tasks in order to reduce the scans’ overheads.

3.2.5 *Measurements*

In this section, we measure CURATOR's effectiveness with respect to a number of relevant metrics. We report results on three different settings: 1) customer clusters, where CURATOR is always turned on, 2) internal corporate production clusters, where CURATOR is also on, and 3) an internal local cluster, where we enable/disable CURATOR to perform controlled experiments.

3.2.5.1 *Customer and Corporate Clusters*

We leverage historical data from a number of customer clusters to assess CURATOR capabilities. In particular, we use ∼50 clusters over a period of two and a half months (June to mid August 2016) to demonstrate CURATOR's contributions to the overall cluster resiliency and data migration tasks. We also collect data from ten internal corporate clusters over a period of three days. These clusters are very heterogeneous in terms of load and workloads, as they are used by different development teams to (stress) test diverse functionalities.

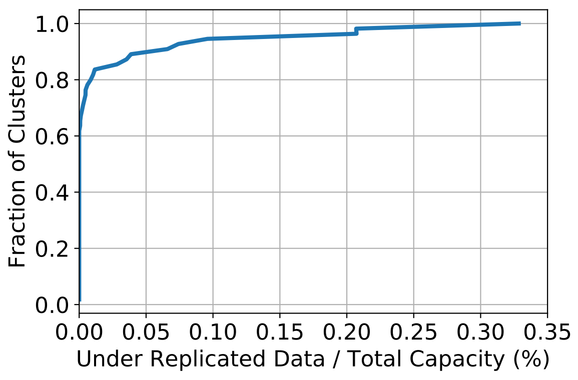


Figure 3.2: Under-Replicated/Total Storage

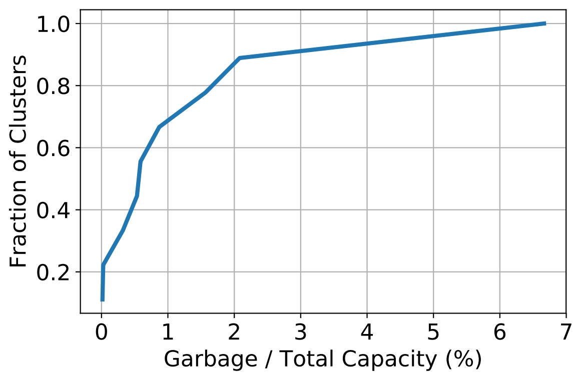


Figure 3.3: Garbage/Total Storage

Recovery Figure 3.2 shows the cumulative distribution function (CDF) of the average under-replicated data as a percentage of the overall cluster storage capacity (in log-scale) in the customer clusters sample. We observe that around 60% of the clusters do not present any under-replication problem. Further, 95% of the clusters have at most an average of 0.1% under-replicated data.

For further confirmation, we accessed the availability cases of the 40% of clusters from Figure 3.2 that reported under-replication. Note that we have access to a database of cases information corresponding to various issues encountered in real clusters, where we can query using different filters, e.g., availability problems, etc. We considered only those cases for the clusters that were opened within 2 weeks of the under-replication event (as indicated by the metric timestamp), and looked for unplanned down time in those clusters. We did not find any unplanned down time in such clusters, which suggests that CURATOR ensured that replication happened upon detecting the under-replication event so that there was no availability loss.

Garbage Collection Figure 3.3 also illustrates the CDF of the (P95) percentage of garbage with respect to the total storage capacity (in log-scale) in the corporate clusters sample. We observe that 90% of the clusters have less than 2% of garbage, which confirms the usefulness of the garbage collection task.

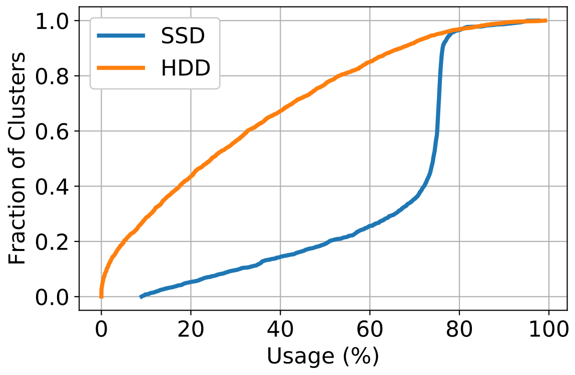


Figure 3.4: SSD and HDD Usage

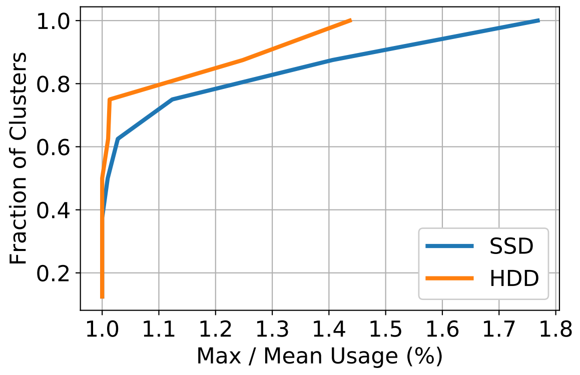


Figure 3.5: Max/Mean SSD and HDD Usage

Tiering Figure 3.4 shows the CDF of SSD and HDD usage in customer clusters. We observe that 40% of the clusters have a SSD usage of at most ∼70-75%. From the remaining 60% of the clusters, many of them have 75% SSD usage, which indicates that the tiering task is doing its job;

the default threshold for tiering is set to 75% in all of these customer clusters, and data has been down-migrated so that the SSDs can absorb either new writes or allow for up-migration of hot data. In the other 10% of the clusters, the SSD usage is slightly higher, which means that although the tiering task is being executed, it cannot entirely cope with such (storage-heavy) workloads. We also note that HDD utilization is typically less, with 80% of clusters having less than 50% HDD usage.

Disk Balancing Figure 3.5 validates disk balancing in the corporate clusters sample. We plot maximum over mean usage ratio, for both SSDs and HDDs. We observe that in 60% (SSDs) and 80% (HDDs) of the cases, the maximum disk usage is almost the same as the mean.

3.2.5.2 Internal Cluster

We are interested in measuring the costs incurred by CURATOR as well as the benefits it provides, with respect to a "CURATOR-less" system, i.e., we want to compare the cluster behavior with CURATOR enabled and when it is disabled. Given that we cannot toggle CURATOR status (ON-OFF) in customer deployments, in this section, we do so in an internal test cluster. We provide a summary of our findings in Table 3.2.

	Metric (Average)	**Curator**	
		OFF	**ON**
Benefits	Latency (ms)	61.73	**12.3**
	Storage Usage (TB)	3.01	**2.16**
Costs	CPU Usage (%)	**14**	18
	Memory Usage (%)	14.7	14.7
	# of IOPS	**1173**	1417

Table 3.2: Benefits/Costs Summary

Workloads We use flexible I/O tester [18] to generate the exact same workloads for testing both settings, i.e., when CURATOR is ON and OFF. We re-image the cluster to the same initial clean state when we toggle CURATOR status.

We simulate three online transaction processing (OLTP) workloads, *small*, *medium*, and *large*, which we execute *sequentially* as part of a single run. Each of these workloads go over three phases, *prefill*, *execution*, and *destroy*. In the *prefill* stage, they create their own user virtual machines (UVMs), together with their associated vDisks. After the *prefill* phase is done, they proceed to *execution*, where the actual workload operations (reads and/or writes) are executed. Following *execution*, the *destroy* stage begins, where the UVMs and associated vDisks are destroyed, i.e., the vDisks' space can be reclaimed.

OLTP Workload	**Data**		**Log**	
	Size (GB)	**IOPS**	**Size (GB)**	**IOPS**
small	800	4000	16	200
medium	1120	6000	16	300
large	1120	8000	12	400

Table 3.3: OLTP Workloads

Each workload is composed of two sections, *data* and *log*, which emulates the actual data space and log writing separation in traditional DBMSes. The *data* section performs random reads and writes, whereas the *log* section is write only. The three workloads only differ on how much data they read/write, and the number of IOPS, as shown in Table 3.3.

Benefits In terms of benefits, we consider latency and storage usage, which mainly highlight the benefits of the tiering and space reclamation tasks. Figure 3.6 shows SSD and HDD usage over time for both CURATOR ON and OFF. We observe that SSD and HDD usage when CURATOR is OFF follows a non-decreasing pattern. When SSDs get full (∼125 minutes), all the data starts being ingested directly into HDDs.

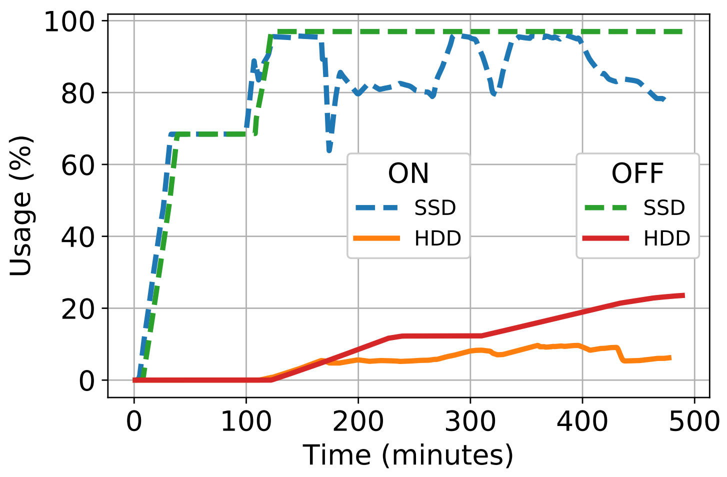


Figure 3.6: SSD and HDD Usage with CURATOR ON and OFF

Instead, when CURATOR is ON, we see the effects of tiering, where colder data is moved to HDDs when the default usage threshold is surpassed (75%). Even though tiering kicks in "on time", the data ingestion rate is so high that the task cannot entirely cope with it, therefore, we observe SSD usage percentages in the 90's. At the end, we see that it reaches the 70's.

Figure 3.6 also illustrates the benefits of garbage collection and data removal tasks. When CURATOR is disabled, we observe a 96% SSD and 23% HDD usage ($\sim$5 TB) at the end of the run, whereas, when CURATOR is enabled, we see a 76% SSD and 6% HDD usage ($\sim$2.27 TB). The average storage usage over the whole run is $\sim$2 TB and $\sim$3 TB for CURATOR ON and OFF respectively. These differences are mainly due to the data removal task. As described above, the *destroy* phase of each workload, where UVMs and associated vDisks are destroyed, allows the data removal task to kick in and start the data removal process, allowing huge storage savings.

Regarding latency, we see an average of $\sim$12 ms when CURATOR is ON, and $\sim$62 ms when is OFF. We measure these values on the *execution* phase of the workloads. As time progresses, the latencies increase when CURATOR is disabled. We speculate this is due to the fact that the newest ingested data goes directly into HDDs, as SSDs are already full, thus, high latency penalties are paid when reads/writes are issued.

Costs Regarding costs, we consider CPU and memory usage, as well as the number of I/O operations performed. From Table 3.2, we see that the number of IOPS executed is higher when CURATOR is ON, as many of its tasks require reading and writing actual data. Still, the overall average IOPS when CURATOR is enabled lies in the same ballpark as the disabled counterpart, $\sim$1400 as opposed to $\sim$1150 when CURATOR is OFF.

We also notice that when CURATOR is ON, the CPU usage is slightly higher. This is due to CURATOR internals, i.e., its MapReduce infrastructure. Although the mappers primarily scan the metadata (mostly I/O intensive), the reducers involve significant logic to process the scanned information (mostly CPU intensive). Even though the average CPU usage is higher when CURATOR is enabled, 18% as opposed to 14%, the value is still in an acceptable range. Regarding memory usage, we do not see a difference between both versions of the system, as shown in Table 3.2.

3.2.6 Reinforcement Learning-based Approach

We have described so far an overview of the distributed storage fabric and delved further into CURATOR's design and implementation, its tasks and policies of execution, etc. In this section, we propose augmenting CURATOR with ML-based scheduling policies. In particular, we leverage the reinforcement learning framework described in Section 2.3, for deciding when to trigger the tiering task. Although our efforts are on the tiering task, our approach generalizes to any of the threshold-based tasks described before.

We start the section with the motivation for the modeling. Then, we delve into more details on the reinforcement learning formulation and the challenges we face when applying RL to our setting and how we address those challenges in our work.

3.2.6.1 Why Reinforcement Learning?

We observed a wide heterogeneity of workloads across the cluster deployments. Given these distinct characteristics of workloads, we noted that generic threshold-based execution policies were not optimal for every cluster, nor for individual clusters over time, as some of them experienced different workloads at different times.

Further, we realized that almost no cluster operator changes the default target threshold of 75% SSD utilization used to trigger the tiering task. For many of the clusters, especially the ones executing mainly *read-like* workloads, this means that they are wasting 25% of fast SSD storage. Thus, in order to efficiently execute CURATOR's tiering task we need to build "smarter" policies that can adapt over time.

The traditional way to improve performance is to use profiling in order to tune certain parameters at the beginning of a cluster deployment. Nevertheless, simple profiling would not easily adapt to the varying loads (and changing workloads) the clusters are exposed to over their lifetimes. We would need to run profilers every so often, and we would discard, in some sense, prior knowledge.

As we do not have a labeled training set, but rather only *partial* information of what happened when executing the tiering task using the sub-optimal threshold-based policy, we need a model that embraces the notion of *exploration*, i.e., try things out to gather information about the world. To make matters worse, moving cold data from SSDs to HDDs typically exhibits *non-immediate* feedback and current decisions may impact performance over a long time, e.g., on subsequent expected or non-expected data accesses. In other words, a future system state in a sequence of states is not necessarily independent of the states that came before it.

We therefore need a model that incorporates both exploration and the ability to work well in settings with non-immediate feedback, where a current decision can have a long-term impact. One such powerful model is reinforcement learning, as described in Section 2.3.

3.2.6.2 State-Actions-Reward

Herein, we outline how we apply reinforcement learning to our problem setting. In order to apply reinforcement learning to schedule storage maintenance tasks, in this case, the storage tiering task, we need to define the set of features that represent the states x, the set of possible actions $\mathscr{A}$, and the reward function.

State We represent states using cluster-level information collected by the cluster management software. Recall that the controller VMs have a comprehensive view regarding cluster state, thus they can be instrumented to provide measurement data. In particular, we use cluster CPU usage, memory usage, storage utilization metrics (e.g., SSD usage), number of read/write I/O operations per second, etc.

Actions We restrict the set of actions to two: either *run* or *not-run* the task.

Reward Regarding the reward function, we use the average cluster latency. As higher rewards are better, though we prefer lower latencies, we actually use negative latencies. The choice of using negative latencies is rather arbitrary, we could have used their reciprocals instead.

3.2.6.3 Generalization and Compression

Given that our state-action space is large, we cannot use a tabular reinforcement learning implementation, as described in Section 2.3. We therefore resort to a function approximator, and not only gain from its compression benefit but also from its power towards generalization.

Many approximators have been studied in the past, such as decision trees [168], neural networks [153, 152], linear functions [58, 207], and kernel-based methods [159]. In this work, we choose linear models. The reason behind this decision is two-fold: 1) we observe that it works reasonably well in practice, and 2) we do not have access to enough training data to bootstrap (or pre-train) more complex models in an offline manner before we deploy our agents—as we shall see next.

3.2.6.4 Faster Training: Bootstrapping

Another key aspect to take into account when training reinforcement learning agents is related to the long time it usually takes them to converge. In other words, it may take a long time before the state space is explored enough so that the agents can start making the “right” decisions. Therefore, we need some strategy to improve their data efficiency if we plan to use them for our storage tiering task in real clusters.

In practice, incorporating prior knowledge, even if incomplete, before the agent is deployed, might help to speed up learning and reduce the amount of interactions with the real environment, which may be limited and costly [15, 156, 122]. Given that we have access to real traces from production clusters, we leverage that already-collected data to pre-train our agents in an offline manner, so as to accelerate training as well as help towards generalization (i.e., the agents would have access to a broad set of data from different clusters). Note that even though we use data sampled from the (sub-optimal) threshold-based policy to “boostrap” our agents, it is still useful to reach “good” states sooner.

3.3 Evaluation

In this section, we evaluate our RL-based scheduler, and compare it to the threshold-based solution currently deployed in Nutanix clusters.

3.3.1 Setup

Cluster We use an internal cluster to run our experiments. The cluster consists of 4 SSDs and 8 HDDs, for a total storage size of $\sim$1.85 TB for SSDs and $\sim$14 TB for HDDs.

Workloads We use flexible I/O tester [18] to generate the workloads. We run the following five synthetic workloads in our experiments:

- *oltp*: performs random reads and writes, 50% reads and 50% writes. Further, 10% of its I/O operations, either reads or writes, have 32k block sizes, and 90% 8k blocks. It has a working set size of 1120 GB, and performs 8000 IOPS. With this workload we intend to simulate a standard database workload.

- *oltp-skewed*: similar to *oltp*, but is *read-only*. It performs 8k block random reads according to the following distribution: 90% of the accesses go to 10% of the data. It has a working set size of 4480 GB, and performs 32000 I/O operations per second. With this workload we aim to better understand the effects of hot data skewness.

- *oltp-varying*: alternates between *oltp* versions that perform 6000 and 4000 IOPS every 20 minutes. In this case, we aim to simulate varying loads of the same type of workload within a cluster.

- *oltp-vdi*: runs the *oltp* workload in one node while the remaining nodes execute VDI-like workloads in 100 VMs each. A VDI-like workload consists of a working set size of 10 GB, with 80% reads and 20% writes. The total number of IOPS per VM is 26. Here, the idea is to simulate (concurrent) heterogeneous workloads within a cluster.

- *oltp-dss*: alternates between an *oltp* version that perform 6000 IOPS and a decision support system (DSS) workload every 20 minutes. The DSS workload is *read-only*, with a working set size of 448 GB. 100% of the reads are sequential, the reads are 1 MB, and the total number of IOPS is 2880. In this case, we attempt to simulate that the workload itself changes over time.

Dataset In order to speed up training in the real cluster, we build a dataset from a subset of the 50 customer clusters introduced in Section 3.2.5 to bootstrap our agent. In particular, we use $\sim$40 clusters, from which we have fine-grained information to represent states, actions, and rewards. The data consists of $\sim$32k transitions, sampled from the (sub-optimal) threshold-based policy. Every cluster was using the same default threshold (75%) for running the tiering task. We split the dataset into training (80%) and test (20%) sets, and do 3-fold cross validation (with grid search) in the training set for hyper-parameter tuning. We standardize the features by removing the mean and scaling to unit variance.

Reinforcement Learning We use Q-learning, described in Section 2.3.1.2, as our reinforcement learning algorithm, and two linear models as function approximators, one for each action. We train them with stochastic gradient descent [33], with l_2 regularization, and the squared loss.

Once deployed, the bootstrapped agent keeps on learning by interacting with the environment. We use the popular ε-greedy strategy, i.e., with probability ε the agent selects a random action, and with probability $1-\varepsilon$ the agents selects the action it currently thinks is the best. We use $\varepsilon = 0.2$ in all our experiments. Further, we set the discount factor $\gamma = 0.9$.

3.3.2 Results

3.3.2.1 Summary

We present a summary of the results for the five different workloads described above in Table 3.4. The experiments are within the order of few hours.

We observe that in all of the cases our Q-learning solution reduces the average latency, from $\sim$2% in the *oltp-varying* workload, up to $\sim$20% in the *oltp-skewed* one, as well as improves the total number of SSD bytes read. We believe that further improvements could also be achieved by adding more features to the states, e.g., temporal features, HDD usage, etc. We also notice that Q-learning demands more IOPS. This is the case since our solution, in general, triggers more tasks than the baseline, thus more I/O operations are performed. Overall, we see that our approach can trade manageable penalties in terms of number of IOPS for a significant improvement in SSD hits, which further translates into significant latency reductions, in most of our experimental settings.

3.3.2.2 SSD Reads

Figure 3.7 shows the evolution of SSD reads for the *oltp* and *oltp-skewed* workloads. We observe that our method based on Q-learning performs on average more SSD reads ($\sim$8 GB and $\sim$24 GB respectively) than the baseline for both workloads, which is a consequence of performing more tiering operations during periods when the offered load from applications is low.

3.3.2.3 Q-learning in Practice

We now provide performance data corresponding to a sample scenario and illustrate how the Q-learning model operates in practice. Figure 3.8 shows the IOPS, latency, and scheduling decisions made while executing the *oltp-dss* workload with our RL-based scheduler. Our system polls the state of the cluster every 30 seconds, if it had not triggered tiering recently, in order to assess

Workload	Metric	Policy	
		fixed threshold	**q-learning**
oltp	Avg. Latency (ms)	12.48	**10.60**
	SSD Reads (GB)	31.68	**39.16**
	Avg. # of IOPS	**2551.54**	2903.20
oltp-skewed	Avg. Latency (ms)	18.55	**14.91**
	SSD Reads (GB)	151.99	**176.28**
	Avg. # of IOPS	**6686.90**	7221.01
oltp-varying	Avg. Latency (ms)	17.28	**16.95**
	SSD Reads (GB)	469.28	**488.17**
	Avg. # of IOPS	**7884.94**	8192.32
oltp-vdi	Avg. Latency (ms)	15.41	**13.92**
	SSD Reads (GB)	40.83	**41.27**
	Avg. # of IOPS	**4450.18**	5178.13
oltp-dss	Avg. Latency (ms)	61.65	**53.00**
	SSD Reads (GB)	4601.17	**6233.33**
	Avg. # of IOPS	**3105.60**	3239.59

Table 3.4: Results Summary

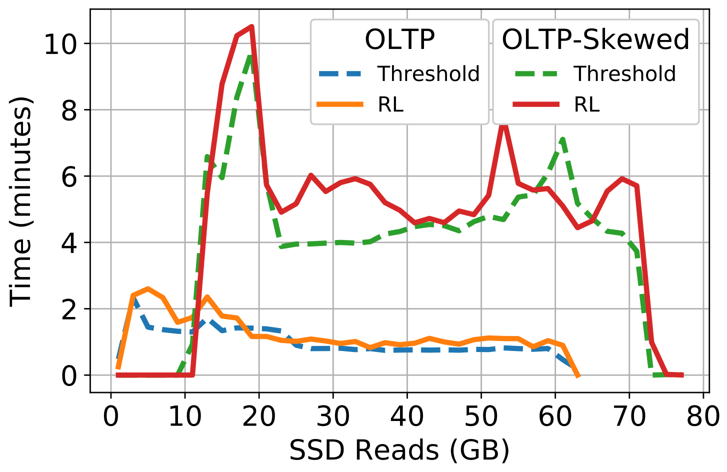


Figure 3.7: SSD Reads in OLTP and OLTP-SKEWED Workloads

whether tiering should be performed. After a tiering task is triggered, we wait for 5 minutes before making a new decision, as we do not want to schedule tiering tasks back-to-back. Regarding the scheduling plot, we not only include the two choices the algorithm makes, but also differentiate whether its decision was due to exploitation (solid lines) or exploration (dashed lines).

The workload keeps on alternating between OLTP and DSS workloads every 20 minutes. It starts with the former, continues with the latter, and so on. We observe that the OLTP workload, in general, demands more IOPS than the DSS one, and also achieves lower latencies (cyclic behavior).

At the beginning, even with high IOPS and low latency, the algorithm thinks that the best option is to trigger tiering (0-20 minutes). When the DSS workload commences (early 20s), the algorithm still keeps scheduling tiering tasks. In this case, it makes more sense as the cluster utilization is not too high but the latency is. Around minute 44, the algorithm explores the state space by not running tiering (dashed orange line that almost overlaps the solid orange ones that follow). Given that this exploration seems to have found a “nice state” with low latency, it considers the best option is to not to run tiering (first chunk of solid orange lines around minute 45). Note that given our 30 seconds polling interval when we do not run tiering, these lines seem to overlap.

At approximately the 47th minute, the algorithm performs an exploration that triggers tiering

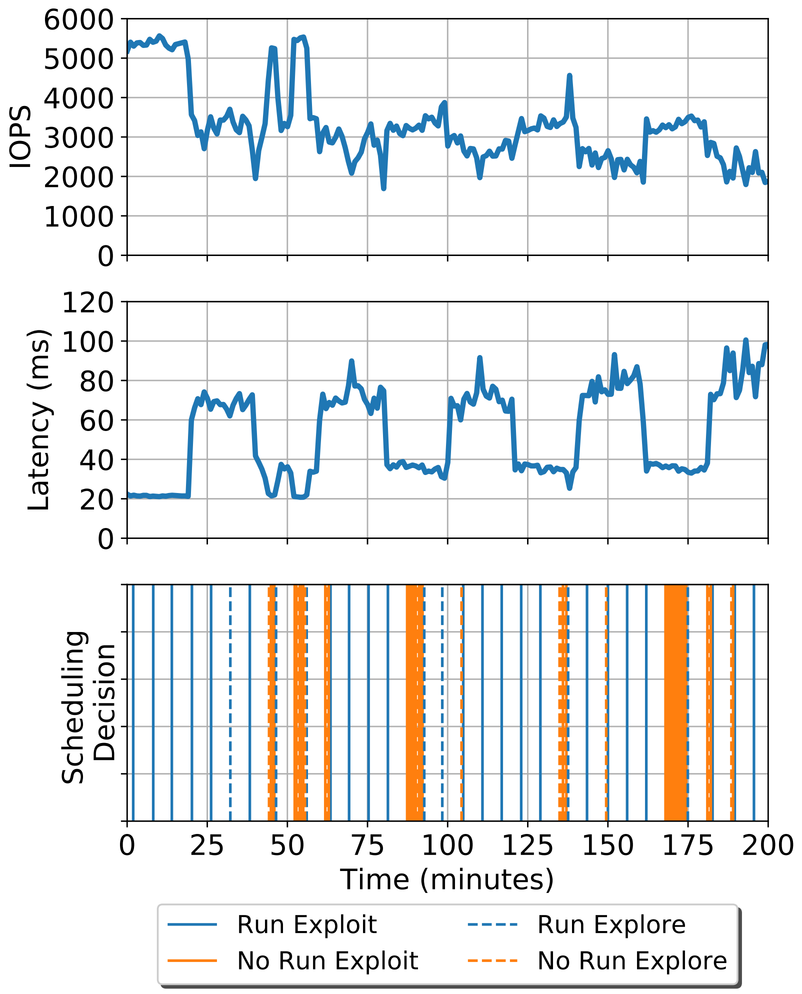


Figure 3.8: OLTP-DSS Workload using Q-learning

(dashed blue line). It does not work out, as later on, the best decisions are still not to run tiering (solid orange lines around minutes 52-54). Around minute 63, when DSS commences again, the algorithm thinks it is best to run tiering. At this point, the cluster is not very utilized, i.e., low IOPS, but the latency is high.

The key thing to notice is that the algorithm seems to be learning that when the cluster is highly utilized (high IOPS) and the latency is low, it should not trigger tiering. During the first period (0-20mins), it was not aware of that, thus it ran tiering, but later on, it started to figure it out (e.g., 40-60mins and 80-100mins periods). Even more noticeable is between the period 160-180mins, where we observe *many* solid red lines (which appears as a single thick one due to the 30 seconds interval). The 120-140mins period is somewhat surprising. We would have expected more solid orange lines there, but they only start appearing towards the end of the period. We believe the algorithm makes early mistakes (minutes 123 and 128), and given that we wait for 5 minutes after running tiering, it can only realize later on ($\sim$133), where it decides that it is actually better not to run.

3.4 Related Work

We present related work in the areas of storage and distributed systems as well as scheduling.

Storage and Distributed Systems CURATOR borrows techniques from prior work on cluster storage and distributed systems, but it composes them in new ways to address the unique characteristics of the Nutanix cluster setting. Note that this setting corresponds to clusters where nodes are heterogeneous and can be equipped with fast storage technologies (SSDs, NVMe, etc.), and unmodified (legacy) client applications are packaged as VMs. Given this setting, the system was designed for client applications to run on the same nodes as the storage fabric, metadata is distributed across the entire system, and faster storage on cluster nodes is effectively used. We now contrast CURATOR [42] with other related work given these differences in execution settings and design concepts.

Systems such as GFS [88] and HDFS [187] are designed for even more scalable settings but are tailored to work with applications that are modified to take advantage of their features (e.g., large file support, append-only files, etc.). Further, they do not distribute metadata, since a single node can serve as a directory server given the use of large files and infrequent metadata interactions. These systems do not take advantage of fast storage—all file operations involve network access and the incremental benefits of fast storage on the server side is minimal.

Cluster storage systems such as SAN and NAS also do not co-locate application processes/VMs with servers. They assume a disaggregated model of computing, wherein applications run on client machines and all the data is served from dedicated clusters [74, 175, 89, 196, 181]. These systems provide scalability benefits and a wide variety of features, such as snapshotting [73], which CURATOR borrows as well. But the crucial points of differentiation are that Nutanix system uses fast local storage effectively through tiering, data migration, and disk balancing. Moreover, we believe that Nutanix solution is the first system to run a continuous consistency checker which results in significant reductions in downtime.

A number of concepts and solutions from distributed systems are used: 1) MapReduce [59] to perform cluster-wide computations on metadata, 2) Cassandra [123] to store distributed metadata as a key-value store, and 3) Paxos [124] to perform leader election for coordination tasks. Interestingly, MapReduce is not a client application running on top of the storage system but rather part of the storage system framework itself.

Scheduling In recent years, there has been an increasing amount of literature on applying machine learning techniques to improve scheduling decisions in a wide variety of areas, such as manufacturing [166, 165, 154, 230, 19], sensor systems [121], multicore data structures [71], autonomic computing [223], operating systems [77], computer architecture [103], etc. For example, in Paragorn [65] the authors propose a model based on collaborative filtering to greedily schedule applications in a manner that minimizes interference and maximizes server utilization on clusters with heterogeneous hardware. Their work focuses more on online scheduling of end-user workloads, whereas ours, concentrates on the background scheduling of cluster maintenance tasks to improve the overall cluster performance.

Wrangler [227] proposes a model based on support vector machines [52] to build a scheduler that can selectively delay the execution of certain tasks. Similar to our task scheduling study, they train a linear model based on CPU, disk, memory, as well as other system-level features, in an offline manner, and then deploy it to make better scheduling decisions. In contrast, our offline trained model only "bootstraps" the reinforcement learning one, which keeps on adapting and learning at runtime, i.e., in an online manner.

Quincy [104] introduces a flexible framework for scheduling distributed jobs. The authors use a graph to frame the scheduling problem, where edge weights encode competing jobs demands (e.g., fairness), and a standard solver computes the optimal schedule according to a cost model. Smart Locks [72] is a self-tuning spin-lock mechanism that uses reinforcement learning to optimize the order and relative frequency with which different threads get the lock when contending for it. They use a somewhat similar approach to our CURATOR study, though they target scheduling decisions at a much lower level.

Perhaps the most similar line of work comes from optimal control [138, 1, 2, 3]. The papers by Prashanth et al. [1, 2] propose using RL for tuning fixed thresholds on traffic light control systems. They propose a Q-learning model that adapts to different traffic conditions (e.g., queue lengths on the lanes, daytime, etc.) in order to switch traffic light signals (green/yellow/red). We use a similar approach but in a different setting, where we tune static thresholds to better schedule data migration in a multi-tiered storage system.

3.5 Summary

Current cluster storage systems are built-in with a wide range of functionality that allows to maintain and improve the storage system's health and performance. In this work, we presented CURATOR, a background self-managing layer for storage systems in the context of a distributed storage fabric used in enterprise clusters. We described CURATOR's design and implementation, its management tasks, and how the choice of distributing the metadata across several nodes in the cluster made CURATOR's MapReduce infrastructure necessary and efficient.

Given the high heterogeneity we observed across clusters, we focused our attention on building smarter task scheduling policies. In particular, we proposed to augment CURATOR with an *ML-based policy* that uses *reinforcement learning* to address the issue of when the management tasks should be executed. Our approach leverages historical traces from real clusters to speed up training and evaluation on simulated workloads in a real cluster achieved up to $\sim$20% latency improvements over a threshold-based approach.

4 | AdaRes

Virtual execution environments allow for consolidation of multiple applications onto the same physical server, thereby enabling more efficient use of server resources. However, users often statically configure the resources of virtual machines through guesswork, resulting in either insufficient resource allocations that hinder VM performance, or excessive allocations that waste precious data center resources. In this work, we first characterize real-world resource allocation and utilization of VMs through the analysis of an extensive dataset, consisting of more than 250k VMs from over 3.6k private enterprise clusters.

Our large-scale analysis confirms that VMs are often misconfigured, either overprovisioned or underprovisioned, and that this problem is pervasive across a wide range of private clusters. We then propose AdaRes, an adaptive system that relies on a *ML-based mechanism* to dynamically adjusts VM resources in distributed virtual environments. Our system uses the *contextual bandits* framework, exploits easily collectible data, at the cluster, node, and VM levels, to make more sensible allocation decisions, and leverages transfer learning to safely explore the configurations space and speed up training.

Empirical evaluation shows that AdaRes can significantly improve system utilization without sacrificing performance. For instance, when compared to threshold and prediction-based baselines, it achieves more predictable VM-level performance and also reduces the amount of virtual CPUs and memory provisioned by up to 35% and 60% respectively for synthetic workloads on real clusters.

In summary, the main contributions of this work are:

- We present a large-scale study of VM resource allocations and usage within thousands of enterprise clusters, which enables us to characterize the overprovisioning, underprovisioning, and variation in resource utilization over time that occurs in this context.
- We propose, design, and implement ADARES, an adaptive system capable of tuning VM resources to increase overall system efficiency that is compatible with existing cluster schedulers.
- Finally, we propose an ML-based mechanism that uses contextual bandits to drive the resource adjustments and leverages transfer learning to speed up training. We instantiate our model with an appropriate formulation that results in more efficient resource allocations in real clusters.

We structure this chapter as follows: Section 4.1 presents the large-scale measurement study of VM resource allocation within hundreds of enterprise clusters, where we show significant overprovisioning, some underprovisioning, as well as important fluctuations of resource needs over time. Then, we describe the design of ADARES, our adaptive system capable of adjusting VM resources on-the-fly, together with our bandits formulation in Section 4.2. We show evaluation results in Section 4.3, related work in Section 4.4, and future work in Section 4.5. We summarize in Section 4.6.

4.1 Resource Utilization Measurements of Enterprise Clusters

This section presents our measurement study on resource allocation and utilization of enterprise clusters with virtual execution environments. Our study characterizes the VM resource allocation problem in the context of enterprise clusters and motivates the need for ADARES.

4.1.1 Measurement Methodology

We perform our measurements on enterprise clusters running the Nutanix commercial virtual execution platform described in greater detail in Section 3.1.

Nutanix cluster manager transparently allocates and migrates VMs based on user configured resource settings and cluster-level utilization metrics. In addition, the platform provides transparent access to highly available virtual storage (virtual disks) located within each cluster node.

Our dataset was collected from sensors deployed on the cluster nodes that record data regarding a broad class of metrics, such as the resources utilized by a VM (e.g., CPU and memory) and cost of various operations (e.g., average I/O latency). The dataset consists of a subset of the clusters that push diagnostic information to a centralized data collection service and refers to the period from April 23^{rd} to May 20^{th}, 2018. Table 4.1 shows an overview of the virtual execution environments that we study, containing more than 250k VM traces.

Metric	**Value**
# of Companies	2,003
# of Clusters	3,669
# of Nodes	17,633
# of VMs	252,941

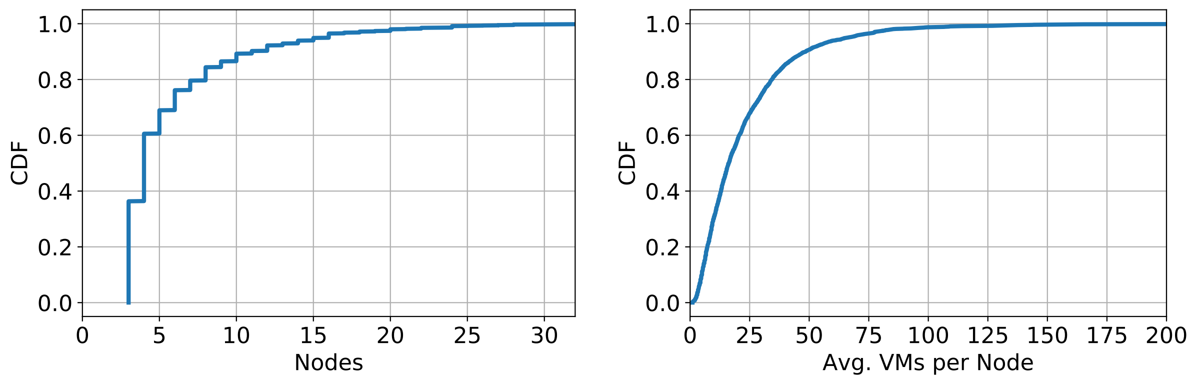
Table 4.1: Dataset Overview

4.1.2 Private Cluster Configurations

To better understand the configuration patterns of enterprise clusters, we perform an analysis of configurations at cluster, node, and VM levels.

Cluster-level Configuration Figure 4.1 shows the distribution of nodes per cluster (4.1a) and the consolidation factor, i.e., the average number of VMs per node, (4.1b). From Figure 4.1a, we observe that 60% of the clusters have 4 nodes or less, and 30% have between 5 and 10 nodes. In general, the clusters have a modest number of nodes. We find that under these environments, when users need additional nodes, companies tend to expand their computational resources by adding clusters, as opposed to adding nodes to existing clusters.

There are three main reasons for this: 1) smaller clusters provide better fault isolation, 2) most of the analyzed clusters are deployed on premise, in remote office/branch office (RoBo) configurations, and 3) some companies prefer to create clusters for each line of business. Figure 4.1b shows that 50% of the clusters have, on average, at most 16 VMs per node, and that 20% have more than 35 VMs per node, up to ∼200 VMs per node.

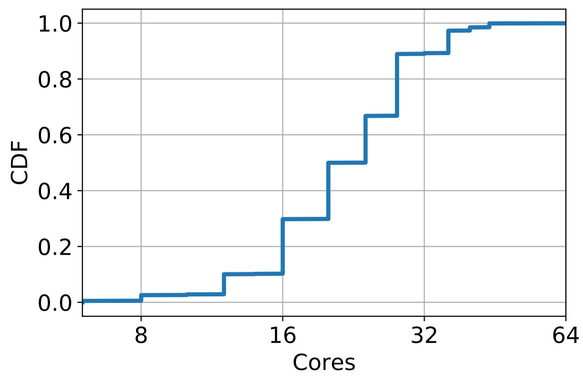


(a) Nodes per Cluster

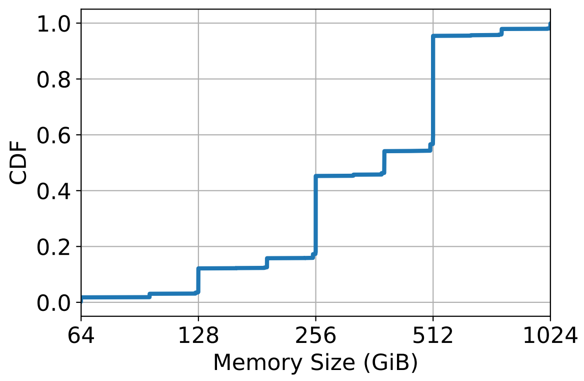
(b) Consolidation Factor

Figure 4.1: Cluster-level Configuration

Node-level Configuration Enterprise clusters often have powerful nodes, as shown in Figure 4.2. We observe that 50% of the nodes have more than 24 physical cores and 384 GiB of RAM, and 10% have at least 36 cores and more than 512 GiB of RAM.



(a) CPU



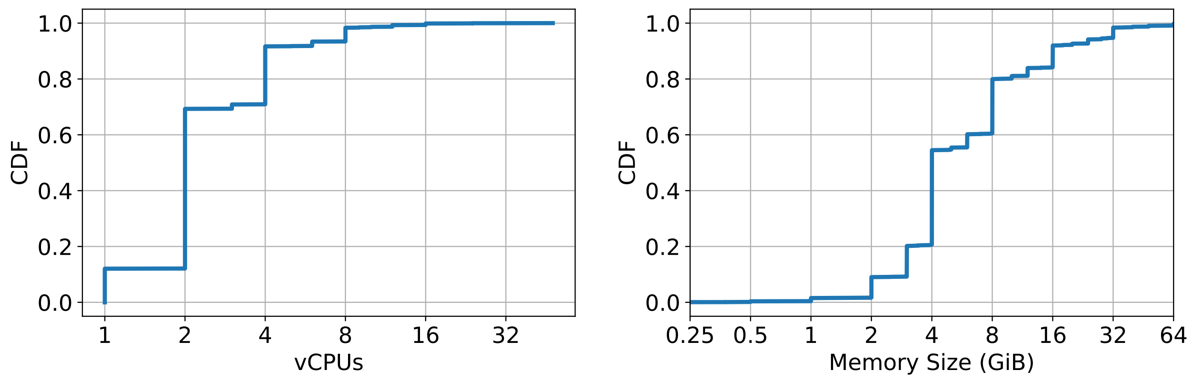
(b) Memory

Figure 4.2: Node-level Configuration

VM-level Configuration Figure 4.3 provides an analysis of the VM sizes in terms of virtual CPUs (vCPUs) and allocated memory. Our dataset shows that approximately half of the VMs are configured with 2 vCPUs, whereas 20% are configured with 4 vCPUs. Regarding memory, around 35% of the VMs are deployed with 4 GiB of RAM, and 20% with 8 GiB. In both resources, we note a "human" sizing pattern of using powers of 2.

We also observe a correlation between the size of the clusters and the number of VMs per node: small clusters have on average the lowest VM density because such clusters typically run a small number of applications supporting limited workloads. In contrast, larger clusters typically support a broad mix of workloads, with some supporting applications such as virtual desktop infrastructure (VDI), which typically deploy a large number of VMs for each connected user. Further, we note that many medium-sized VMs (i.e., VMs with 2-4 vCPUs) are typically used to deploy server applications such as SQL Server, MS Exchange, etc.

Summarizing, enterprise clusters are often small-sized single-tenant clusters, with powerful nodes, that support the workload requirements of small and medium-sized businesses.



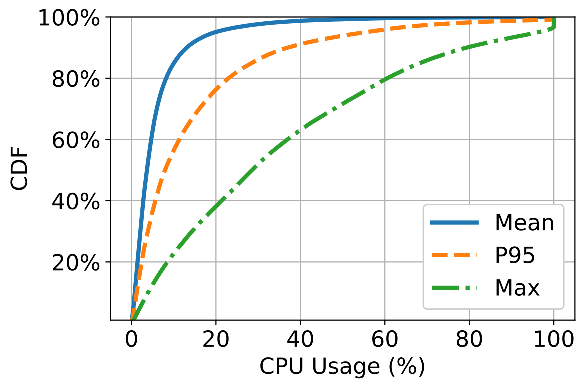
(a) CPU (b) Memory

Figure 4.3: VM-level Configuration

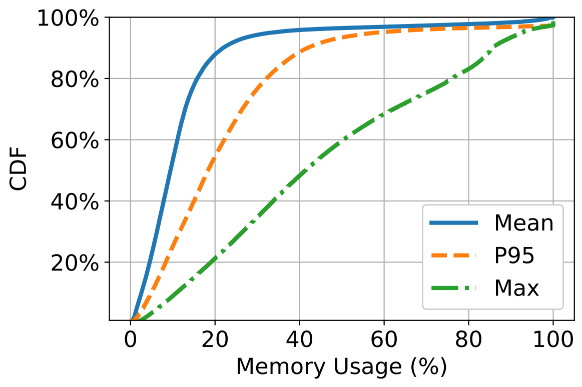
4.1.3 Problem Characterization

This section provides an analysis on the utilization of the clusters. Our analysis relies on several key metrics that we collect and are representative of the VMs resource usage. For each metric, we record, on each cluster node, the average measurement over a 5-minute interval at the VM-level. This data enable us to calculate the mean, maximum, and the 95^{th} percentile (P95) of a series of 5-minute measurements for any given metric.

Figures 4.4a and 4.4b present the cumulative distribution function (CDF) of the mean, P95 and maximum VM resource usage for CPU and memory. These results show that many of the VMs are *overprovisioned* with respect to both CPU and memory. In particular, 90% of the VMs have P95 CPU and memory usages lower than 40%. Further, 80% of the VMs have a maximum resource usage that is lower than 60% (CPU) and 80% (memory) throughout their lifetime; in other words, 40% and 20% of the allocated resources are *never* used by 80% of the VMs. By analyzing the dataset we calculate that the global resources allocated but never used correspond to 26% (CPU) and 27% (memory) of the total allocated resources by all VMs. Intuitively, the areas to the right of the maximum line in Figure 4.4a and 4.4b represent the global wasted resources that are never used, but our numbers additionally take into consideration the different absolute sizes of the VMs. Such



(a) CPU

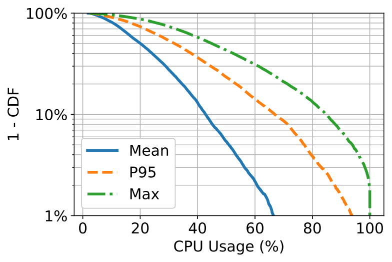


(b) Memory

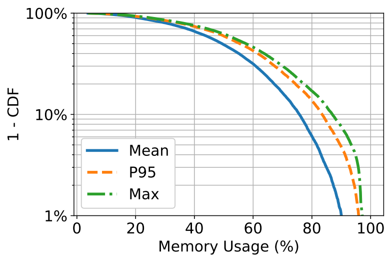
Figure 4.4: VM Resource Usage

allocated but sparsely used resources are the result of two main factors: 1) manual VM resource allocation, and 2) users inability to accurately predict the resource demands of their workloads.

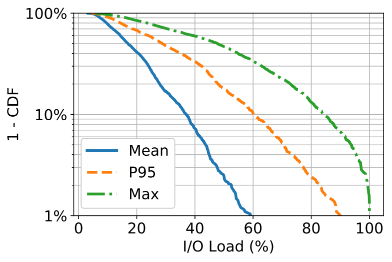
We observe a similar trend at the node level, i.e., many nodes have low average utilization but experience high peak resource usage. We show the complementary cumulative distribution functions (CCDF or 1-CDF) of node-level usage in Figure 4.5. Note that CCDFs are useful for highlighting the tails of distributions. Besides CPU and memory usage, we also analyze the compute processing load of the storage controller on each node and use it as a proxy of the node's



(a) CPU



(b) Memory

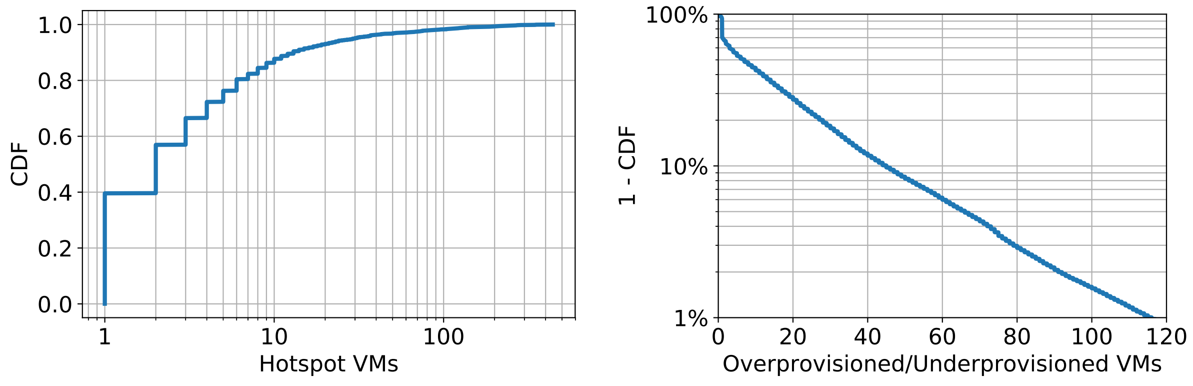


(c) I/O

Figure 4.5: Node Resource Usage

I/O load. In general, we see that node usage is higher than VM-level usage, especially memory utilization, due to oversubscription, where around 10% of nodes have, on average, more than 80% memory usage, but still, many nodes are underutilized.

Although average utilization is generally low, our data still reveals that many VMs are underprovisioned. Figure 4.6a shows the distribution of hotspot VMs per cluster. We consider a VM to be a hotspot if its 95^{th} percentile metric utilization is greater than 75%. We observe that 40% of the clusters with hotspot VMs have at most 2 underprovisioned VMs, whereas 10% of the clusters with underprovisioned VMs contain at least 10 hotspot VMs. From the total clusters in the dataset, 45% contain either CPU-hotspot VMs, memory-hotspot VMs, or both. Thus, our data suggests that underprovisioning is not limited to few, possibly incorrectly managed, clusters; instead, our data reveals that the hotspot problem impairs a large fraction of clusters.



(a) Distribution of Hotspot VMs per Cluster (b) Ratio of Over/Underprovisioned VMs per Cluster

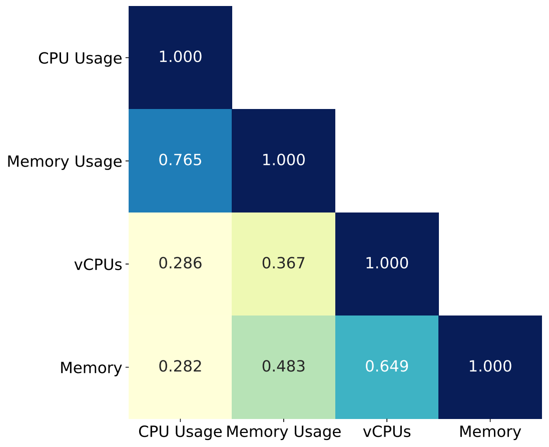
Figure 4.6: Hotspots and Over/Underprovisioned VMs Ratio

Summarizing, most VMs in today's enterprise clusters are not sized appropriately, with many VMs either overprovisioned or underprovisioned. This motivates the need for developing an automated system to determine VM resource allocations as opposed to relying on user-provided configurations.

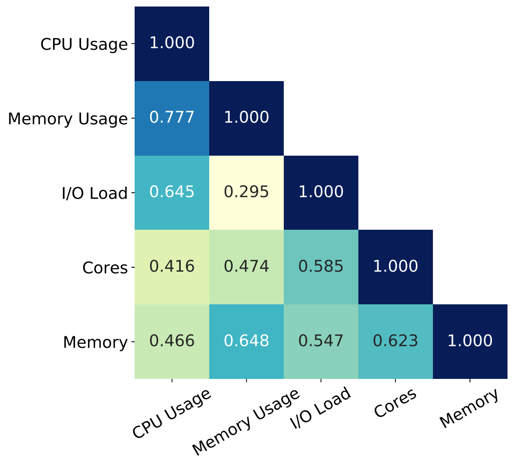
4.1.4 Opportunities and Challenges for Adaptive Resource Allocation

This section highlights some of the challenges and opportunities for adaptive resource allocation. Figure 4.6b shows the distribution of the ratio of overprovisioned divided by underprovisioned VMs (when such underprovisioned VMs exist) per cluster, at a given point in time. We consider a VM to be overprovisioned if its 95th percentile metric utilization is less than 25%. Recall that underprovisioned (or hotspot) VMs are those with a P95 utilization greater than 75%. In general, when there are hotspots, there are also VMs with overprovisioned resources at the same time. For example, we observe that 50% of the clusters with underprovisioned VMs have at least a 7:1 overprovisioned/underprovisioned VMs ratio.

We also correlate the VM/node provisioning and utilization metrics using Spearman's correlation [193], which assesses monotonic relationships between variables (linear or not). We use P95 values of each VM for this analysis. We show the results in Figure 4.7 as a heat map, which intuitively can be interpreted as follows. If metric x tends to increase when y increases, the correlation coefficient is positive. If x tends to decrease when y increases, the correlation is negative. A zero correlation indicates that there is no tendency for x to either increase or decrease when y increases.



(a) VM-level

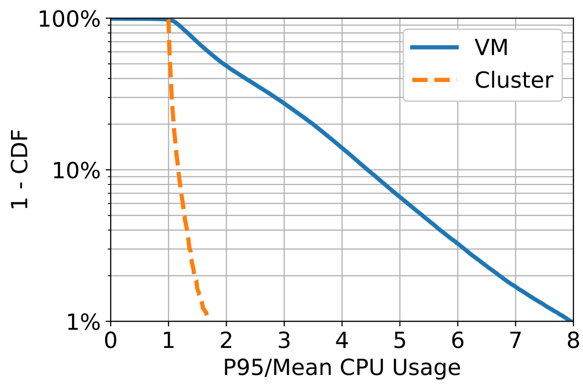


(b) Node-level

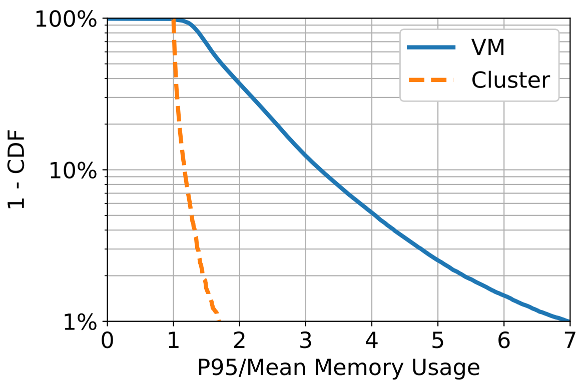
Figure 4.7: Provisioning and Utilization Metrics Correlations

A perfect correlation of ± 1 occurs when each of the variables is a perfect monotone function of the other. We observe that CPU and memory usage have a strong positive (but not perfect) correlation, which seems to indicate that the compute-heavy workloads in our dataset are also memory-intensive, but VM-specific tuning is still necessary to determine how much memory should be provided to a VM to go with the amount of CPU resources allocated to it. Further, the node-level I/O usage is not that strongly correlated with memory and CPU usage, indicating that there is an opportunity to co-locate VMs that are just I/O intensive with VMs that are memory or CPU-intensive.

Next, we examine the variation in resource utilization across time. The purpose of this analysis is to quantify the need for reallocating resources across VMs within a cluster and to examine the implications of static thresholds. Figure 4.8 shows the CCDF of the 95^{th} percentile divided by the mean of CPU (4.8a) and memory (4.8b) usages for both VMs and clusters. We notice that $\sim$45% of the VMs have a P95 at least $2\times$ bigger than the mean, for both metrics, which indicates that there is significant variation across time for many VMs. However, at a cluster-level, the variation of CPU and memory usage over time is insignificant, indicating that usage spikes are not highly correlated across VMs.



(a) CPU Usage



(b) Memory Usage

Figure 4.8: P95/Mean Usage Ratios

Summarizing, many clusters have both underprovisioned and overprovisioned VMs. In fact, there is significant disparity between the utilization levels of VMs in a cluster, regardless of the resource type. This disparity, in turn, provides an opportunity to reallocate resources from VMs that are overprovisioned onto VMs that are underprovisioned, potentially solving both problems. However, such a mechanism would have to address two important challenges: 1) it can only reallocate resources between VMs running at a given time, and 2) it has to continuously adapt to the current load given the large temporal variations in VM resource usage.

4.2 System Design

This section describes the design of ADARES, a system that proposes an ML-based mechanism to change the physical resources allocated to VMs based on workload characteristics and other attributes of the virtual execution environment. Our system crucially relies on the contextual bandits framework to guide the resource adjustment. This section starts with a high-level description of the goals that determined the design of ADARES, an overview of the system, and a description of its core components.

4.2.1 Goals

ADARES is designed to identify the appropriate resource allocation settings for VMs in enterprise clusters. The goal is to improve cluster execution efficiency by allocating the optimal amount of resources to each VM but without compromising VM performance; that is, the resources allocated to a VM should be just adequate for it to operate without experiencing a slowdown. Thus, ADARES reduces the resources allocated to overprovisioned VMs and increases resources allocated to underprovisioned ones.

Note that the VM assignment problem is orthogonal and is out of the scope of this work, i.e., ADARES does not determine the optimal node to which a VM is assigned or migrated to; instead, it relies on existing tools, such as VMware's vShpere/vMotion [184], to address this challenge. Nevertheless, by optimally setting the resource allocation, ADARES allows such tools to both pack more VMs into clusters as well as migrate VMs to the appropriate nodes that have sufficient resources to host them [177].

We design our system with the goal of achieving the following properties:

- *Highly adaptive:* The system should work in a diverse set of operating conditions and identify optimal operating points for a diverse set of cluster, node, and VM configurations. It should continuously adapt VM configurations in response to changes in workloads. Our choice of contextual bandits is primarily driven by its ability to learn and adapt to such settings.

- *Safe allocations:* A key challenge with using the bandit framework in our setting is that the adaptive controller might require a significant amount of unsafe exploration to distill a decent model of cluster behavior. We seek to build a system that can transfer the knowledge gained from simulations and thereby safely streamline the model distillation process in real clusters.

- *Modular and configurable:* Our system should provide a configurable framework that can integrate a variety of measurement sensors and operate using configurable prediction models. Further, we desire a framework that can integrate system management policies defined by the cluster operator (e.g., ensuring that VMs never exceed a certain amount of utilization for a given resource). Moreover, the approach should be general enough to be able to work with many hypervisors and virtualization environments.

4.2.2 Components

This section provides an overview of our system and introduce its core components. Figure 4.9 shows a high level overview of its architecture on top of Nutanix's distributed storage fabric (DSF). ADARES is composed of five core components to optimize VM configurations: the Sensing Service component is deployed on each node in the cluster and runs within the controller VMs (CVMs), whereas the remaining components are executed within the cluster manager node.

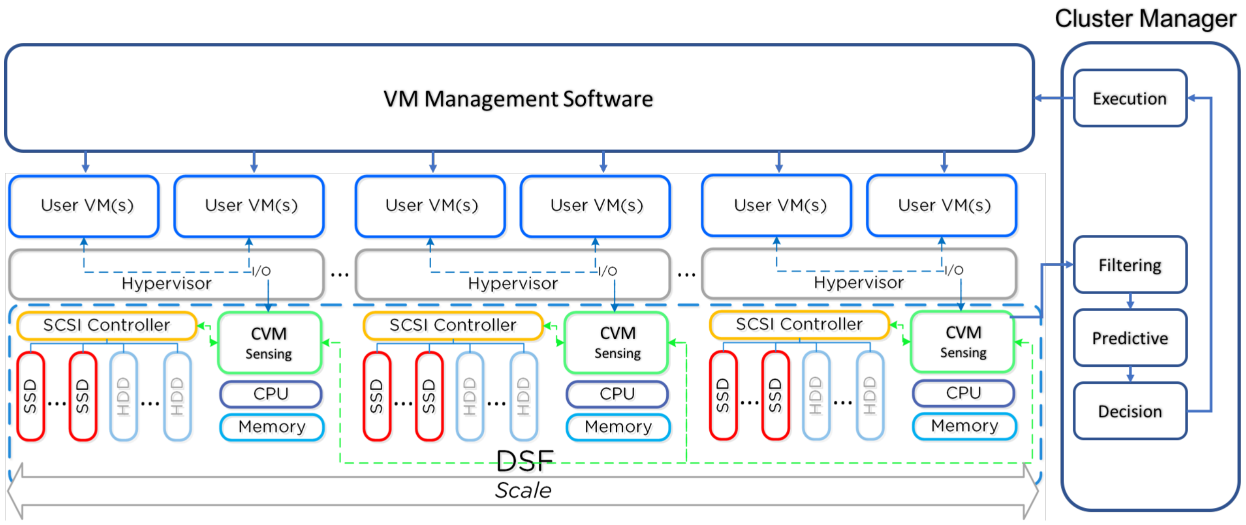


Figure 4.9: ADARES Architecture

4.2.2.1 Sensing Service (SS)

The Sensing Service is in charge of collecting telemetry data. The current version collects data at cluster, node, and VM-level. It utilizes sensors on each of the nodes in the cluster to continuously collect information regarding the utilization levels of resources as well as some key performance metrics of the VMs. For instance, it collects information on the CPU and memory utilization of VMs and the number of IOPS performed by each VM, as well as performance metrics such as CPU ready times, virtual memory swap rates, and the latency of I/O operations. These sensors are typically deployed on the controller VMs (CVMs) running on each node, which not only have access to VM-level metrics (e.g., CPU or memory utilization), but also interpose on I/O operations performed by the VMs on the virtual disks exported by the cluster software.

4.2.2.2 Filtering Service (FS)

The Filtering Service component serves as a preprocessing step running on the cluster manager node and is designed to limit the number of VM configuration changes made by the system at a given time. It enables the operator to filter the collected telemetry data based on different strategies.

For instance, the FS can filter VMs with CPU usage greater than a certain threshold, randomly select a percentage of the total VMs in the cluster, etc. The output of this service is typically a subset of VMs that will be tuned in a given round of the contextual bandit algorithm. As such, the FS component functions as a throttling mechanism, as it can control the rate at which changes are made. This is especially important for highly loaded systems, where changing a large number of VMs at the same time could be counterproductive. It is worth noting that although this component aims to filter inputs in order to avoid unnecessary computations, ADARES can be configured to also discard outputs—as we shall see next in the Decision Service.

4.2.2.3 Predictive Service (PS)

The Predictive Service along with the Decision Service encompass the core contextual bandit logic in ADARES. At a high-level, a machine learning (ML) model identifies the appropriate *arms* or *actions* (e.g., scale up/down a VM's memory allocation), given the current *context* or *state* of the VMs in the cluster (e.g., utilization level and other metrics). The actions are chosen based on some expected *reward*, i.e., the effect of taking the actions on the VM performance metrics. We discussed these concepts in greater detail in Section 2.2.

PS exports two methods as part of its interface: 1) *predict*, which outputs the recommended actions for the selected VMs based on the ML model trained to maximize the expected reward, and 2) *learn*, which supports updating the ML model in an online fashion, in order to fold in the actual observed rewards as a consequence of pulling arms (or taking actions).

It is worth noting that our framework is somewhat agnostic to the specific ML model chosen (e.g., linear models, decision trees, neural networks), as well as the prediction task (e.g., classification, regression), and it could be used outside the contextual bandits domain. For example, the user could potentially train a model to predict the peak CPU utilization of a VM in the next hour and use this prediction to determine whether to increase/decrease the number of vCPUs, or simply design a classifier to decide whether to scale up/down the memory of a VM based on the current performance metrics, without taking into account any expected reward, as opposed to how bandits work.

4.2.2.4 Decision Service (DS)

The Decision Service component makes the final decision regarding changes to resource allocations. PS gives hints to DS (e.g., with high confidence PS can recommend to scale down the vCPUs of a particular VM), but it is up to the DS service to follow PS's advice. DS can be seen as a component that leverages the ML-based predictions, but additionally, folds in two other considerations when determining the actual decisions performed by the cluster manager: 1) *exploring* the configuration space to discover the rewards associated with a diverse set of actions, and 2) *leveraging domain knowledge* to make more sensible decisions given the application domain.

For the latter consideration, DS enables users to configure different rules, such as min-max (hard) bounds of utilization and resources, as well as update levels of resources per VM (or group of VMs). For example, a user could set a configuration to ensure that VDI VMs can only have between 1 and 4 vCPUs, and 2-8 GiB of memory, and that the system must *always* scale up the vCPUs of those VMs if their CPU usage is more than 90%. Further, on every scaling operation the user can configure, for example, to limit the number of updates of vCPUs to $\pm$ 1 and memory to $\pm$ 40%. This feature allows ADARES to be more cautious or aggressive in accordance with the workload resource tolerance.

4.2.2.5 Execution Service (ES)

The decisions made by DS are handed to this service, which triggers the adjustments. In order to perform provisioning changes on-the-fly, the underlying guest OS kernel needs support for hot addition/removal of CPUs and memory. We use Linux guests that provide such support.

Our current prototype supports integration with VMware vSphere,[1] which acts as the VM management software layer, and we use VMware ESXi as the nodes' hypervisor. As VMware vSphere only provides native support for hot addition of both resources but not removal, we use other vSphere APIs to perform the adaptations. In particular, we use APIs to execute programs directly on guests using the VMware Tools agent installed on the VMs, as the resources addition/removal

[1]For details refer to https://www.vmware.com/products/vsphere.html.

can be done with native Linux programs (echo and grep) [55, 145]. Finally, this component also keeps track of the execution progress and notifies the main controller of any failures during the process.

4.2.3 Bandit-based Approach

We now describe how ADARES uses contextual bandits for the VM resource allocation problem. We begin by describing the rationale behind the bandits choice. We then outline how we apply it to our problem setting. Importantly, this section identifies the challenges in using contextual bandits and how ADARES addresses them.

4.2.3.1 Why Contextual Bandits?

Training a model offline using any supervised learning algorithm would not work in our case because VM workloads change frequently and many incoming VMs do not have historical records at all. Such approach would require re-training the model with a high frequency to try to keep up with workload changes and its unclear how often this process would be required to attain acceptable results.

Instead, using an online learning algorithm is more suitable because it automatically and dynamically adapts to new patterns as new data becomes available. One can think of an online model trained to predict workload characteristics of VMs. For example, given a new VM context, a model would predict its maximum CPU usage in the next hour, and if it is above certain target threshold, then the system would scale its vCPUs up. However, even if we had a perfectly accurate predictive model, we would not have an easy way to properly fold the result of taking the action into the model, as the prediction task is decoupled from the result of the action. Furthermore, the action taken would have affected the actual max CPU usage of the VM during the hour, complicating the learning process.

We therefore need an online formulation where the learning task itself could estimate the result (i.e., reward) of taking an action, given side information (i.e., VM context). As we do not know *what would have happened* had we taken a different action, our model should take different actions so as to refine its estimates. The two main models that encompass the above characteristics are contextual bandits and reinforcement learning, described in greater detail in Chapter 2.

Reinforcement learning (RL) [199, 201, 198] is oftentimes seen as an extension of the contextual bandit setting. One difference is that the reinforcement learning agent can take many actions until it observes a reward. For example, in a chess game, the player makes many moves but the reward is only revealed at the end of the game (win, loss, draw). This sparsity makes the problem harder to learn and gives rise to the so-called credit assignment problem, i.e., which actions along the way actually helped the player win? Further, in RL, a current decision may have an impact over a long horizon.

Although in our setting RL would be ideal, we can make certain simplifying assumptions to make the problem easier to learn in practice, and therefore be able to model it using contextual bandits. We do not need to deal with sparse rewards; after we scale a VM, we can sense its performance metrics with our Sensing Service and get an idea of how much the scaling action affected the VM. But most importantly, given the high number of changes we perform to VMs, we can assume that a current change will not have an impact on the VM performance over a long horizon (e.g., on the next day).

Even though recent successes in deep reinforcement learning [153, 152] have made it quite popular among practitioners, most RL algorithms lack theoretical guarantees. On the contrary, there are many contextual bandits algorithms with strong theoretical guarantees that ensure convergence to an optimal solution [131, 6, 28], and they typically have a faster ramp up than their RL counterparts—another important aspect towards a successful practical implementation.

4.2.3.2 Context-Actions-Reward

In order to apply contextual bandits to manage VM resources, we need to define the set of features that represent the contexts x, the set of possible actions $\mathscr{A}$, and the reward function. Crucially, all this setup depends on how the rest of the system is structured, as in what can be measured and how the performance of an application VM can be quantified.

Context We represent the context of VMs by cluster, node and VM-level features, as well as temporal information. The context attributes include the various measurements collected by the Sensing Service, e.g., the resource allocations made to VMs, current and historical resource utilization levels (at VM, node, and cluster granularities), summary statistics of those (e.g., max, min, average, and P95 utilization), performance metrics that characterize VM behavior (e.g., latency, IOPS, swap rates, CPU ready time, etc.), overcommitment factors of the node and cluster where the VM is running, and others. Is worth noting that the ability to feed side information into the agent, allows the agent to do context-dependent adaptations, and makes the whole contextual bandits framework well-suited for our setting.

The intuition behind including global information, i.e., cluster and node-level features besides just the VM information, is to aid the agent in making more “coordinated" scaling decisions across VMs, by also taking into account availability of resources in the host(s), oversubscription levels, etc. For instance, when the side information shows that a node's resources are highly overcommitted, the agent might decide not to increase the resources of its VMs. Or when it detects sinusoidal usage patters in VMs, it may decide to augment and decrease their resources depending on the part of the cycle it is in, and so on.

Actions We use a special case of the general contextual bandit framework introduced in Section 2.2, in which the action set $\mathscr{A}_t$ remains unchanged for every round t. In particular, we define a total of three actions per resource type (*scale up*, *scale down* or *noop*). For example, the agent can choose to scale up memory and scale down vCPUs, scale down both, neither, etc. Actions result in resource allocations updates to VMs, and in turn, VMs respond to the new allocations by exhibiting an updated set of utilization and performance metrics, which the agent then uses to update its model.

Reward The final step in setting up the bandit formulation is to define the reward function. The primary objective in defining the reward function is to steer the cluster configurations towards states that correspond to minimal VM-level resource allocations without compromising VM performance. Our framework is agnostic to the way the reward function is defined; the only constraint it imposes is that the reward must be a function of the various metrics gathered by the Sensing Service.

We give a reward of 1 when, irrespective of the action, we move from a “bad” state to a “good” one, e.g., from a context with swapping and/or CPU overload to a context without. We also give a payoff of 1 if we make “good” actions, e.g., if we scale down to increase the usage, but the VMs do not end up incurring in swapping or CPU overload, or if we scale up to try to escape from a state with swapping or high CPU load. On the other hand, we penalize (zero reward) actions that lead to bad states, e.g., if we are not swapping and after scaling down we start doing so. Finally, we also penalize scaling up/down recommendations of PS if the domain knowledge encoded in DS heuristics (i.e., hard bounds) don’t allow them.

We note that there are likely many formulations of the reward function that achieve the desired objective of maximizing system efficiency without hurting VM performance. We plan to provide the cluster operator with the ability to configure the reward function by incorporating additional information from application-level performance metrics, as that would allow for more precise reward valuations and faster convergence to optimal configurations.

4.2.3.3 Safe Allocations and Faster Training: Sim2Real

Another challenge of applying bandit-based approaches in our setting is that we need to ensure reasonable performance and respect “safety” constraints during the learning process. We need to be extra cautious not to mess up with VMs while exploring different actions but, at the same time, we want to make the right decisions as soon as possible. Incorporating “prior knowledge” before the agent is deployed might help to speed up learning and reduce the amount of (costly) interactions with the real VMs [15, 112].

Inspired by the robotics community, as well as prior work on the systems space [79], herein, we build a cluster simulator to pre-train our agent. The idea is to then transfer the knowledge gained while training on this (cost-less) simulator to *bootstrap* our agent before it is deployed in real clusters. We start the section by stating what we need from the simulator, the challenges its construction presents, and how we address those challenges in our work.

Requirements The simulator should provide an easy mechanism to emulate, to some extent, the dynamics of a cluster. We are interested in modeling what happens to VM performance metrics once we perform configuration changes. In other words, we need (simplistic) analytical models of the environment that our Sensing Service can query to obtain the contextual information (or features) and rewards necessary to train our agent.

Challenges Although we brought robotics into the picture, building a simulator of a robot is a completely different endeavor. Therein, the well-defined rules of physics (e.g., gravity) aid in the otherwise even harder process. Herein, we don't have those; the large number of components and connections (e.g., VMs, nodes, storage devices, queues), the intricate dependencies (e.g., hypervisors multiplexing shared resources), and the irregular resource needs (e.g., different workloads changing over time) complicate our ability to create a simulator that faithfully represents a real cluster.

Nevertheless, from a machine learning standpoint, we don't need an entirely "accurate" simulator, we need a reasonable initialization of what we believe the dynamics are, and then we can keep updating those beliefs as we keep on training in the real cluster. By incorporating (incomplete) initial knowledge, the agent would be exposed to the relevant regions of the context and action spaces from the earliest steps of the learning process, thereby eliminating the time needed in random exploration for the discovery of these regions, as in safe reinforcement learning [85].

Data-driven approach Following the "reasonable" premise above, we use a data-driven strawman approach to build our cluster simulator. We run a set of controlled experiments on synthetic workloads that aim to mimic the ones we observe in real clusters, and we perform different changes to VM configurations and record their impact. For example, we change the amount of vCPUs assigned to VMs and observe how those changes affect their CPU usage.

Further, we run different I/O benchmarks using vdbench [212] to profile IOPS and latencies for different representative workloads (e.g., 8k random reads, 8k random writes, 1M sequential writes, 8k 50% random reads and 50% random writes, burst, sequential) at different rates, and with different outstanding I/O per node. This profile data gives us an idea of the rates at which our system can (roughly) serve the different types of I/O.

Given that we have an estimate of the service rates, and as we know the amount of outstanding I/O in a node, we then resort to queueing theory (single server model or M/M/1) to compute arrival rates per node, and then derive approximate latencies (or wait times) in the system. Finally, we also create multi-queue multiprocessor schedulers with round robin per node, to roughly estimate CPU ready times among the VMs running in those nodes. We acknowledge that the addition of extra features to the simulator can (and probably will) get us better results on real clusters. We leave that to future work.

4.2.4 Controller

Having introduced the core constructs of ADARES, and the instantiation of the contextual bandits approach, in this section, we show how we use our system together with the latter framework to dynamically adjust VM resources.

The core services described in Section 4.2.2 are orchestrated by a controller running in the cluster manager node. Listing 1 shows a (simplified) example of the main controller loop, the heart of our agent. The agent starts sensing the cluster state (cluster, node, and VM-level information). Note that in our setting we define contexts $x_t \in \mathbb{R}^d$ per VM, thus here $X_t \in \mathbb{R}^{nxd}$, where n is the number of VMs in the cluster, and d the size of our feature vector. The i^{th} row in matrix X_t represents the context of the i^{th} VM.

The agent then uses FS to select b VMs eligible for allocation updates in the current round, where $b \leq n$, and contacts the Predictive Service to obtain the recommendations for those filtered VMs ($P_t \in \mathbb{R}^{bx|\mathscr{A}_t|}$, where $|\mathscr{A}_t| = 9$ is the number of possible actions) (Line 4). In this and the next step is where the bandit algorithm comes into play. After obtaining the predictions, the Decision Service uses an exploration/exploitation strategy together with its domain knowledge to decide which actions to take ($A_t \in \mathbb{R}^{bx1}$, i.e., only one action per VM).

Listing 1 ADARES Controller

```
1:  $X_t \leftarrow$ ss.sense(cluster) (sense context)
2:  for $t = 1,2...$ do
3:      $X_t \leftarrow$ fs.filter($X_t$) (filter VMs)
4:      $P_t \leftarrow$ ps.predict($X_t$) (get prediction values)
5:      $A_t \leftarrow$ ds.decide($P_t$) (explore/exploit + domain knowledge)
6:      es.execute($A_t$) (execute actions)
7:      $X_{t+1} \leftarrow$ ss.sense(cluster) (sense new context)
8:      $R_{t,A_t} \leftarrow$ reward($X_t$, $A_t$, $X_{t+1}$) (compute rewards)
9:      ps.learn($X_t$, $A_t$, $R_{t,A_t}$) (online learning)
10:     $X_t \leftarrow X_{t+1}$ (update context)
11: end for
```

The set of actions are passed to ES for the actual execution (Line 6). After the actions are executed, the agent uses the Sensing Service to get a sense of the actions' impact on the VMs performance metrics. Note that $X_{t+1} \in \mathbb{R}^{nxd}$, i.e., we sense the whole cluster, not just the previous b filtered VMs. We do this because we will use these new contexts in the next iteration (Line 10), and because the filtering step (Line 3) may select a different subset of VMs than in previous iterations. The agent computes the rewards *only* for the b filtered VMs of the current round. Finally, the agent learns the benefits/drawbacks of taking actions A_t for contexts X_t in Line 9.

4.3 Evaluation

We implemented ADARES in about 7.8 kLOC of Python. Our current prototype is built in the context of the same Nutanix commercial virtualization product that we used to collect the cluster measurements. In this section, we present the evaluation of our prototype with experiments on real clusters.

4.3.1 Setup

Cluster We have full control over an experimental cluster. This mainly homogeneous cluster consists of a total of 48 cores, a CPU capacity of 115.2 GHz and 512 GiB of RAM, on which we run around 20-36 VMs.

Virtualization Software We use VMware ESXi 5.5.0 as the hypervisor, and our Execution Service talks to vSphere to change the virtual hardware associated with the different VMs. We generate VM images with CentOS Linux 7, kernel version 3.10.x.

Further, we clone the VMs in our experiments from the three instance types shown in Table 4.2. None of the VMs can have less than 1 vCPUs and 2 GiB of RAM, but their maximums differ based on the type. Finally, we set the same tuning aggressiveness for all VMs, $\pm$ 1 for vCPUs and $\pm$ 512 MiB for memory.

VM Instances	**Resources**			
	Initial		**Min-Max**	
	vCPUs	**Mem (GiB)**	**vCPUs**	**Mem (GiB)**
large	2	3.75	1-4	2-7.5
xlarge	4	7.5	1-8	2-15
2xlarge	8	15	1-16	2-30

Table 4.2: VM Instance Types and Min-Max Ranges

Workloads We simulate different workloads using a modified version of flexible I/O tester [18], where we can configure the VM CPU load, the workload active memory size, and the I/O operations per second. We attempt to mimic the real workloads we observe in our traces, some VDI-based workloads, other Server-like workloads (e.g., SQL server), etc. We mainly issue 8k block-sized I/O. Depending on the workload, we do random reads, random writes, and both random reads and writes (50% each, 70-30%, or 80-20%).

Methods We use the following methods in our experiments:

- *passive*, where no configurations adjustments are done to VMs. This is the baseline currently deployed in Nutanix clusters.

- *reactive*, where we sense information about the VMs and if their usages are above/below certain threshold(s) we perform the adaptations.

- *proactive*, similar to *reactive*, but uses a machine learning model to predict maximum usages sometime in the near future (e.g., 10 minutes). It performs changes if the predicted utilization levels deviate from the configured target threshold(s).

- *bandits*, our method, where we adjust resources using contextual bandits.

We use 75% as the underprovisioned threshold for the *reactive* and *proactive* baselines; that is, if the current or predicted VM resource usage (either CPU or memory) is above 75%, the system scales the resource(s) up. Similarly, we use a 25%-threshold to indicate overprovisioning, i.e., if the current or predicted VM resource usage is below that threshold, we scale the resource(s) down. Our system makes decisions every 5 minutes.

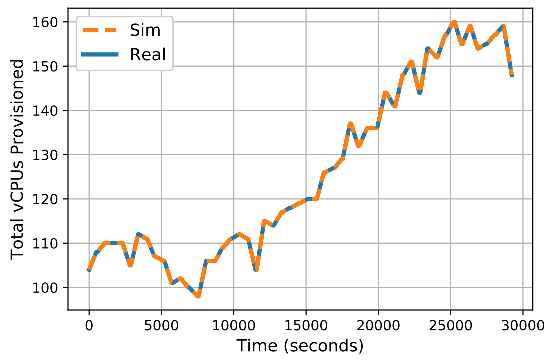
Machine Learning Further, we use two linear models, one for each resource, to predict the max utilization of each resource in the next 10 minutes, in the *proactive* baseline. We train the models using stochastic gradient descent [33], with l_2 regularization, and the squared loss. We use the default hyperparameters of scikit-learn [162].

For our method, *bandits*, we use LinUCB, the popular upper confidence bound (UCB) [205] algorithm described in Section 2.2.1.2. We set the exploration constant to 0.5 (higher means more exploration), and the regularization parameter of the ridge regression to 0.01.

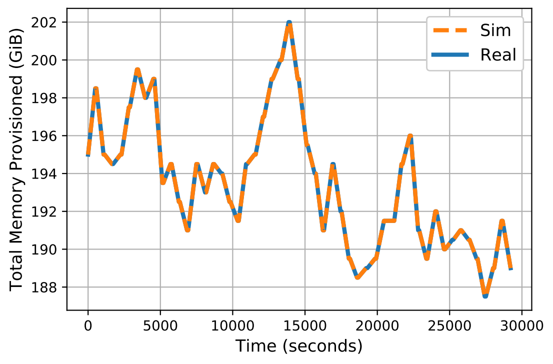
4.3.2 Results

4.3.2.1 Cluster Simulator Fidelity

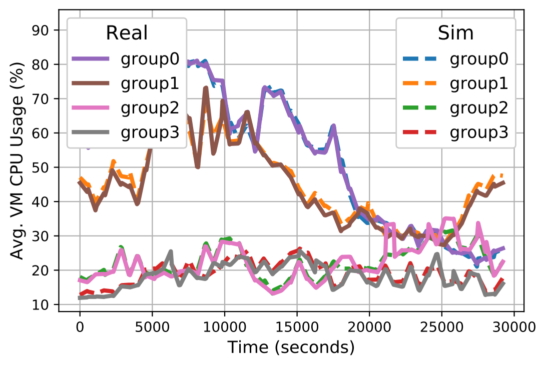
We start off by evaluating the fidelity of our cluster simulator. Herein, we instantiate 26 *xlarge* VMs in our cluster. We group them in four distinct groups, each with a different workload pattern and I/O intensity. We perform random configuration changes during a 8-hour period. We record all the actions done along the way, and we then replay those exact same actions in our simulator.



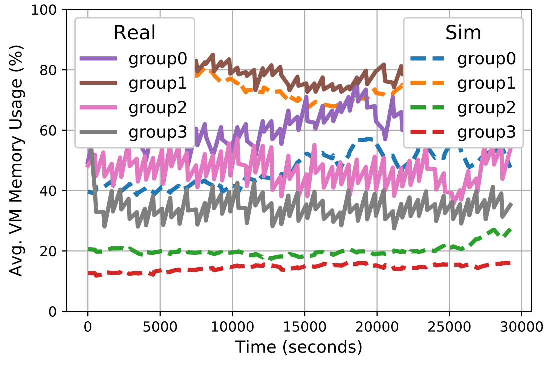
(a) Provisioned vCPUs



(b) Provisioned Memory



(c) CPU Usage



(d) Memory Usage

Figure 4.10: VM Resource Provisioning and Utilization

Figures 4.10a and 4.10b show the total vCPUs and memory provisioned across the VMs over time. Both the simulator (Sim) and the real cluster (Real) lines overlap, as we are replaying the same actions in the simulator. More interestingly, Figure 4.10c shows the average VM CPU usage across the four different VM groups. We observe that the simulator is doing a pretty good job in estimating the CPU usage of all the groups when we perform adaptations. Similarly, Figure 4.10d illustrates the memory usage across groups. We note that our simulator mostly underestimates the usage, which is most notoriously for groups 2 and 3. However, it seems to follow the line trend (e.g., groups 0 and 1) but is off by some constant factor.

Finally, Figure 4.11 shows the average VM latency decomposed in groups doing random reads (RR) and random writes (RW). We see that our simulator does a better job at estimating RWs operations, though it also does a decent job for random read I/O.

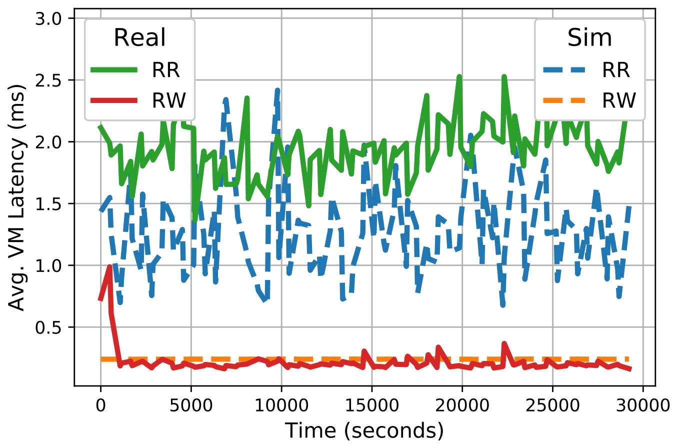
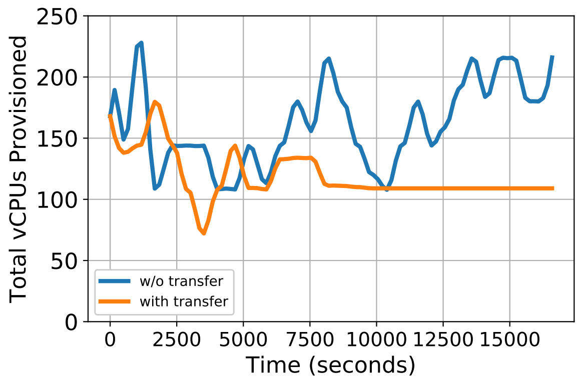


Figure 4.11: Latency

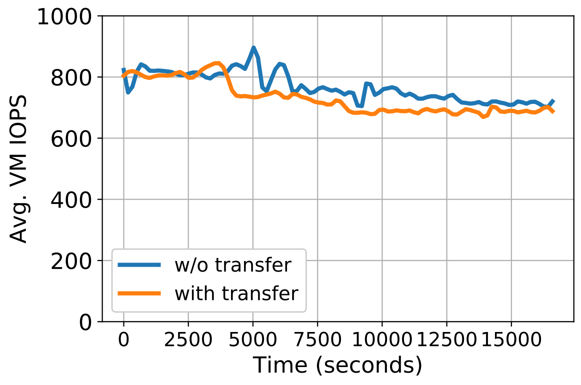
4.3.2.2 *Transfer Learning:* Sim2Real

In this section we evaluate how transfer learning helps to speed up training in real clusters. We run different static workloads across a set of 36 VMs, 12 of each of the instance types described in Table 4.2, for a period of ∼4 hours. Herein, we compare the two flavors of our bandit-based approach, with and without transfer learning. Note that we pre-train in our simulator using VMs that run other workloads in order to avoid overfitting. Still, if we were running the same workloads and overfitting, it would be an extra evidence of the reasonable performance of our simulator.

Figure 4.12a shows the total vCPUs provisioned over time for both bandit-based approaches, with and without transfer learning. We observe that the allocations are much more stable when we pre-train. Transfer learning lead us to a $2\times$ saving of vCPUs allocations for this workload (109 vCPUs as opposed to 216). Even more, without pre-training, the agent ends up allocating more vCPUs than the ones it started with. This latter statement highlights the importance of safe exploration while applying these type of methods. Figure 4.12b shows the average I/O operations per second of the VMs in this workload. We observe that, even though we saved $2\times$ vCPUs, we are still able to perform very close to the vanilla bandit version in terms of IOPS.



(a) Provisioned vCPUs

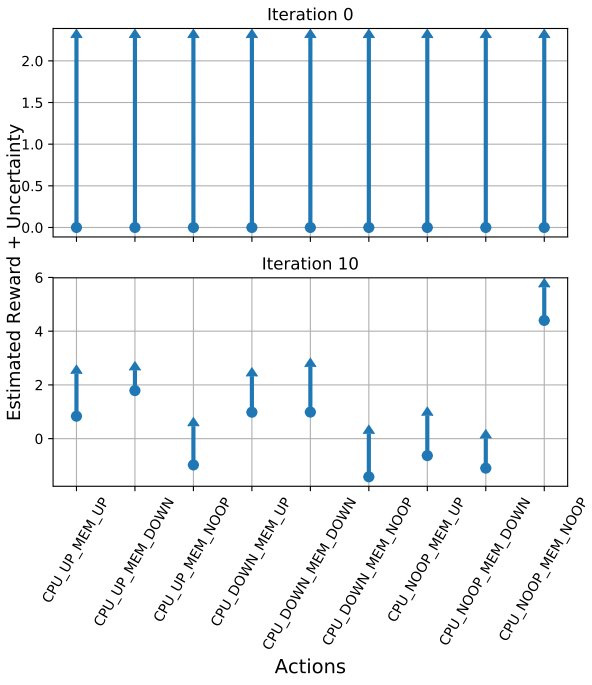


(b) VM IOPS

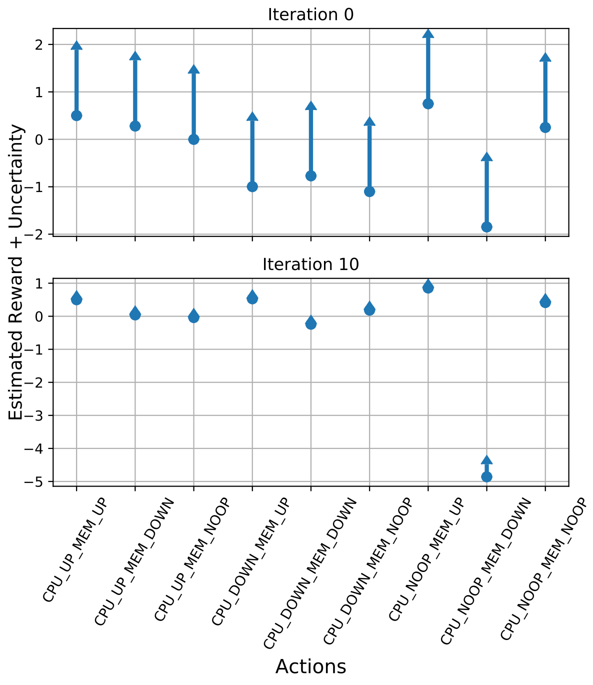
Figure 4.12: vCPUs Allocations and VM IOPS

We now illustrate how transfer learning helps LinUCB to accelerate training. Figure 4.13 shows the estimated reward and uncertainty of the different actions for a random VM context that has memory underprovisioning. We observe that the estimated rewards start at zero (solid dots) and uniform uncertainty (long lines with caps), when we start training from scratch (top of Figure 4.13a). As the agent learns, the confidence bounds shrink for that same context. However, the agent still recommends to do nothing CPU_NOOP_MEM_NOOP, the action with highest score.

On the other hand, Figure 4.13b shows the benefits of “bootstrapping" our model. At the top, we see non-uniform confidence bounds. Note that the agent is able to recommend the right action for this context (CPU_NOOP_MEM_UP), from the beginning, due to the knowledge transfer. Few iterations later, the upper confidence bounds are close to the expected reward, and the leading actions are still the ones that involve scaling up memory.



(a) Without Transfer Learning



(b) With Transfer Learning

Figure 4.13: LinUCB and Transfer Learning

4.3.2.3 Workloads

Static Herein, we evaluate static workloads, which are characterized by a somewhat flat utilization profile over time. To that end, we use the same setting as Section 4.3.2.2, where we run workloads on a set of 36 VMs, 12 of each instance type, during 4 hours. We only report results on the *bandits* version that uses transfer learning.

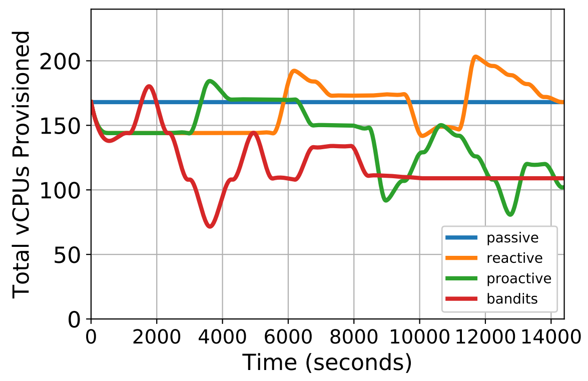
Figure 4.14a shows the vCPUs allocations over time for the different methods. We see that both *proactive* and *bandits* result in the fewest allocations, although our method converges to a steady state sooner.

Further, Figure 4.14b plots the CDF of CPU usages of VMs, both at the beginning and at the end of the runs. We observe that around 30% of the VMs start with 100% CPU usage, and $\sim$35% are using less than 20% of their computational resources. As expected, the initial curves have an almost perfect overlap, as every method runs the same workload. More interestingly, at the end of the runs, we can see how the adaptive methods increase the usages of overprovisioned VMs (by scaling them down), as well as decrease the usage of underprovisioned ones (by scaling up). For example, in the *bandits* method, 35% of the VMs have at most 55% of CPU usage, and only less than 10% of the VMs have 100% CPU usage, as opposed to the initial 30%.

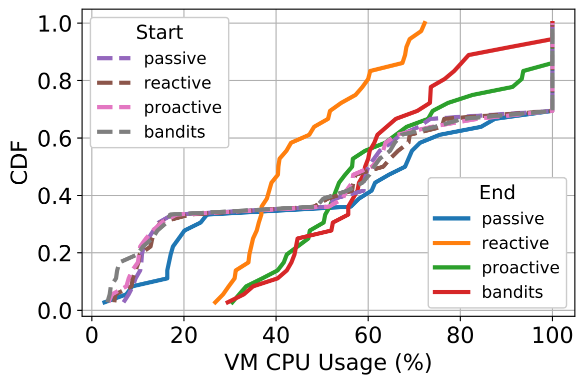
Overall, we see a 35% improvement, in terms of amount of vCPUs allocated, for the ML-based methods (*bandits* and *proactive*), when compared to static or threshold-based approaches. Further, at the end of the run, the standard deviation of the VMs CPU usage is 18% and 22% for *bandits* and *proactive* respectively, as opposed to 35% of *passive*, i.e., a 48%-37% improvement. Although the deviation of *reactive* is lower (14%), the average VM CPU utilization also is, 46% as opposed to 62% of *bandits*.

Increasing Another example of workloads we observe in practice are those with increasing resource demands. In this case, we simulate a workload with increasing working set size (WSS). We augment the WSS every 20 minutes for a group of 20 *xlarge* VMs running in our controlled cluster.

Figure 4.15 shows the results of 4-hour runs. From 4.15a we observe that both *reactive* and *proactive* begin by hot removing memory from VMs. Around 6k seconds, the sensed memory usage goes above 75%, thus *reactive* starts scaling up. The surprising fact is the *proactive* allocations do not change. By looking at the predictions from this method, we observe that it always predicts a maximum memory utilization less than 75%, therefore, it does not perform adaptations. We speculate the reason is that it has not enough information to start making "accurate" predictions yet.



(a) Provisioned vCPUs



(b) VM CPU Usage (Start-End)

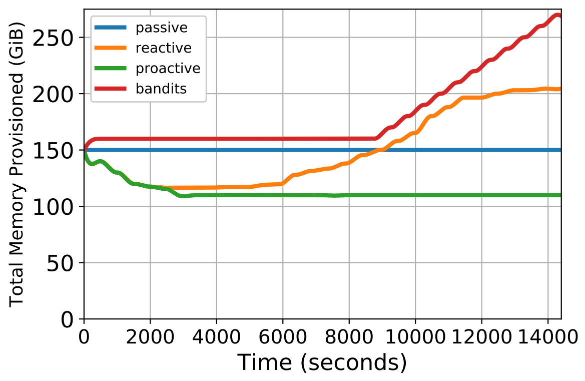
Figure 4.14: Static Workload

Bootstrapping the method with our simulator using the same idea of transfer learning could have helped.

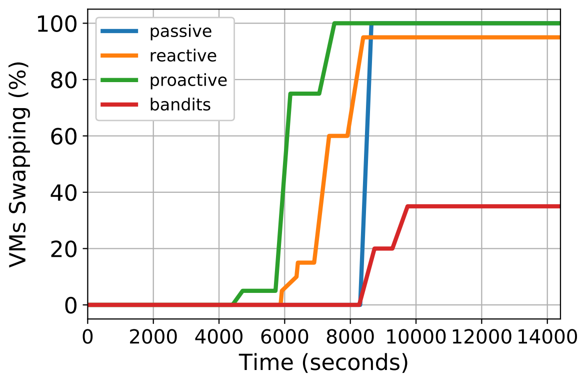
On the other hand, we can see that the *bandits* method allocates slightly extra memory than *passive* during the initial ∼130 minutes of the run. As the agent starts receiving punishments (or zero rewards) because of increasing swapping levels in the guest OSes, it starts scaling up (around 9k seconds). This phenomenon can be observed in Figure 4.15b, where we show the percentage of VMs that experience swapping over time. As expected, *bandits* performs the best, as it is being trained to avoid such states (or contexts).

Further, Figures 4.15c and 4.15d compares the average cluster latency and the total cluster IOPS of *passive* and *bandits* methods. We observe that our method shows lower I/O latency in general, and it can keep up with the workload IOPS. Overall, if we consider the number of VMs that are experiencing swapping at the end of the run, we can see *bandits* has a 63-65% improvement over the other baselines.

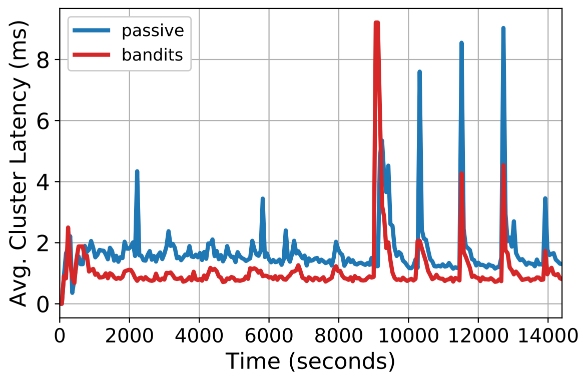
Periodic and Static We now focus on periodic and static workloads. In particular, we vary CPU utilization levels of VMs (Figure 4.16a), but keep constant the memory usage (Figure 4.16b). We



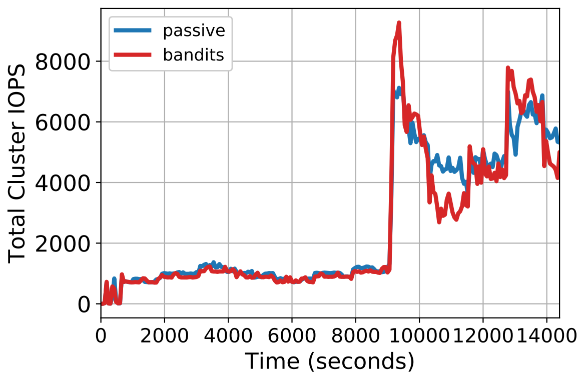
(a) Provisioned Memory



(b) VMs doing Swapping



(c) Cluster Latency

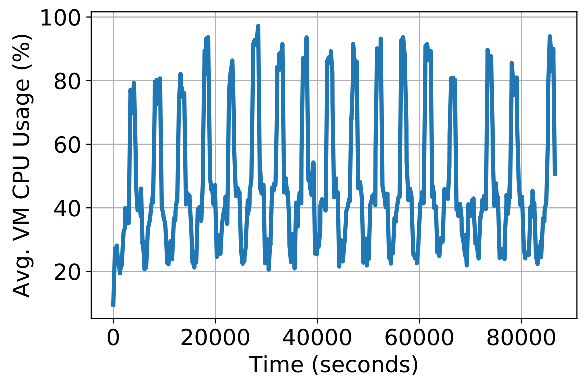


(d) Cluster IOPS

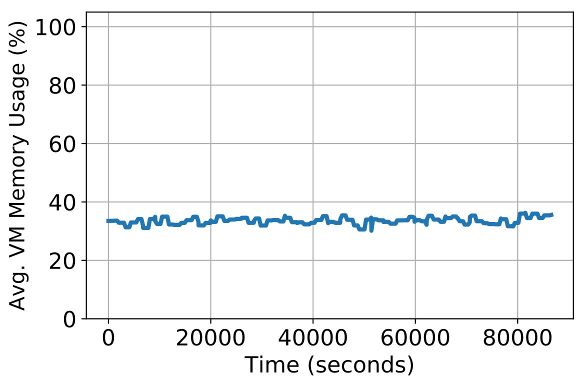
Figure 4.15: Increasing Memory Workload

expect the adaptive methods to decommission CPU resources during non-peak times, and restore them back during high demand, and also, reduce the amount of provisioned memory to increase utilization. We run a 1-day long experiment, where we deploy four AdaRes agents and execute them in parallel, one for each method. Each agent controls 8 *xlarge* VMs, which are evenly spread across the nodes in the cluster.

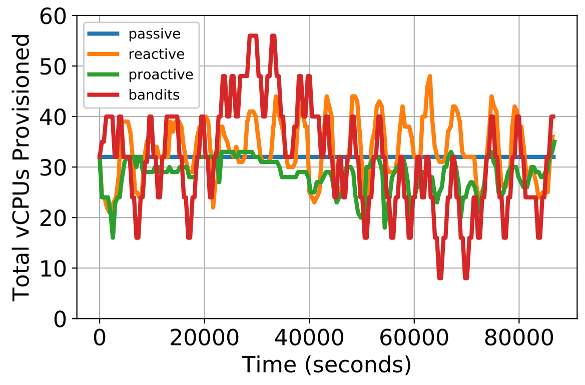
On average, we observe that the ML-based adaptive methods provision less vCPUs and memory than the other two (Figures 4.16c and 4.16d), which translates into higher resource utilization (Figure 4.16e). For example, the average memory usage of *bandits* almost doubles *passive*'s usage.



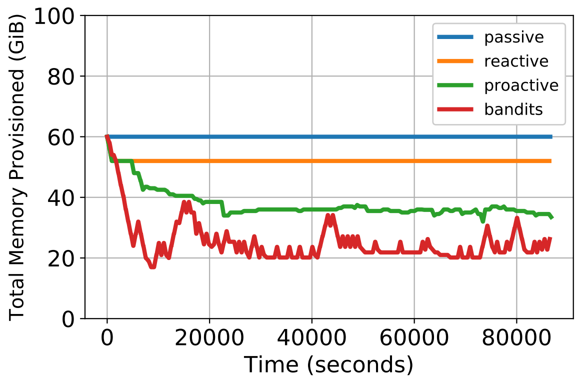
(a) CPU Pattern



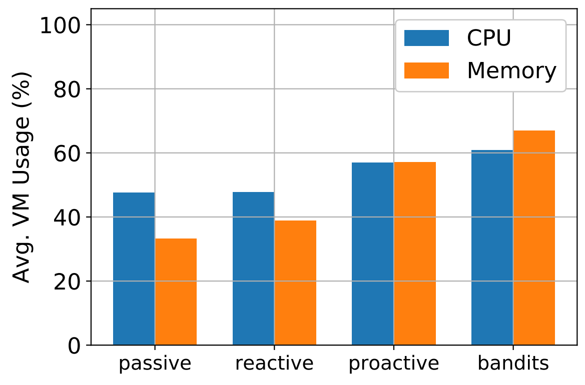
(b) Memory Pattern



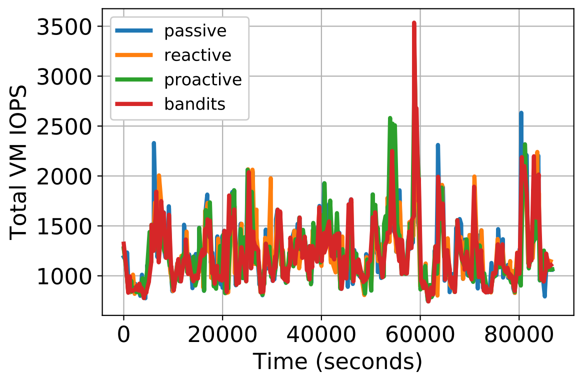
(c) Provisioned vCPUs



(d) Provisioned Memory



(e) VM Usage



(f) VM IOPS

Figure 4.16: Periodic and Static Workload

Further, even though *bandits* uses less resources, it can still keep up with the IOPS of the workload (Figure 4.16f). Overall, *bandits* ends up using around 25 GiB of RAM, almost a 60% reduction over the static baseline, while at the same time keeps reducing the CPU overload during peak times. Finally, we acknowledge that different threshold settings can cause completely different behavior for the *reactive* and *proactive* approaches. Even for *bandits*, hyperparameter tuning of exploration constants, more advanced feature engineering, or non-linear models (both for *bandits* and *proactive*), could boost these numbers up. We leave that to future work.

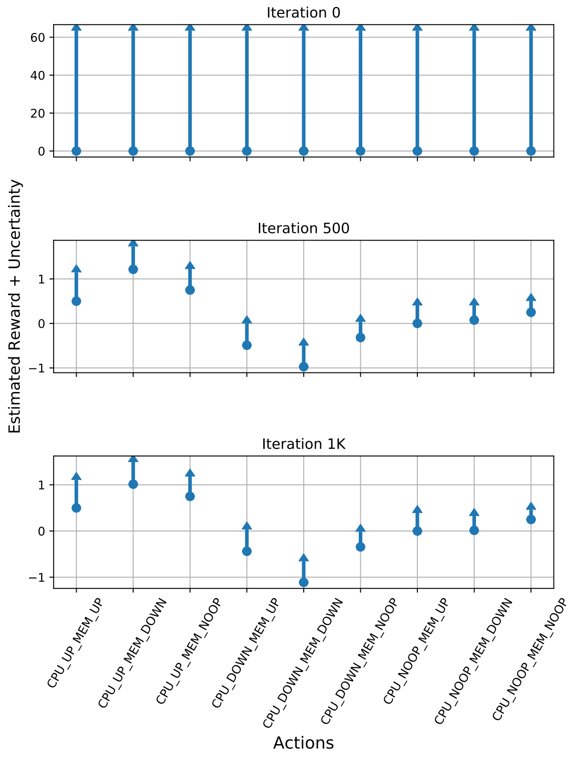
4.3.2.4 LinUCB in Practice

Finally, we illustrate few more examples of how LinUCB operates in practice. We use our cluster simulator to replicate the real cluster environment, and we run heterogeneous workloads across 36 VMs during 1k iterations. We checkpoint our model every 500 iterations to be able to track the progress.

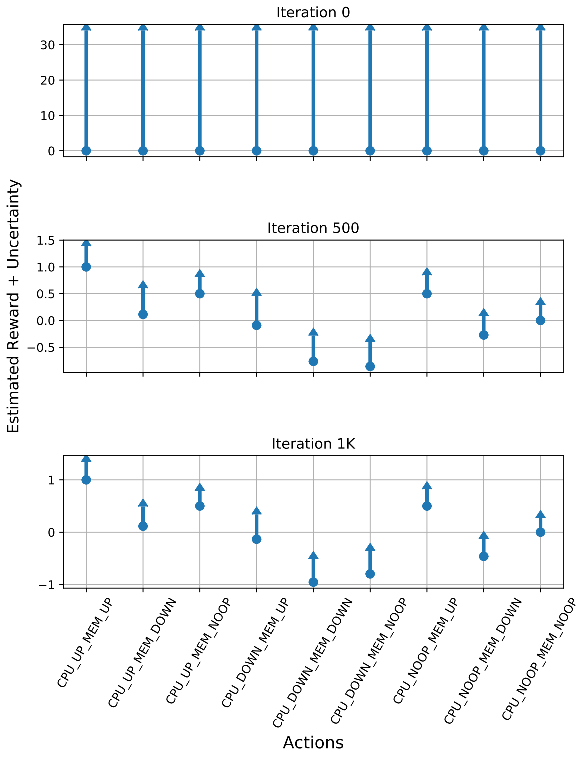
We randomly select a VM with CPU underprovisioning and one with both CPU and memory underprovisioning. We show the estimated reward and uncertainty of the examples in Figures 4.17a and 4.17b respectively. In both cases, we observe that the estimated rewards start at zero and there is high uncertainty in every action. As the algorithm performs exploration, those intervals shrink and the estimated rewards get closer to the expected rewards for each action (Iteration 500). The algorithm then starts exploiting and choosing the actions with the highest expected reward. From Figure 4.17a, we see that the “best” actions are the ones that involve scaling up vCPUs, as the VM experiences high CPU overload. Although the *noop* action seems to be the most explored one, as its confidence interval shrinked the most, its estimated reward is still below the aforementioned actions. On a similar note, Figure 4.17b illustrates that scaling up both vCPUs and memory is the clear winner for VM contexts with underprovisioning of both resources.

4.4 Related Work

We discuss work relevant to ADARES [44] in the areas of measurements and different approaches towards resource management (RM).



(a) CPU Overload



(b) CPU Overload and Memory Swapping

Figure 4.17: Different Contexts

Measurements: Google traces [173, 224] have enabled research on a broad set of topics, from workload characterizations [149] to new algorithms for machine assignment [172]. However, they characterized a month-long trace of non-VM workloads. In this work, on the other hand, we focus on VM workloads running in enterprise clusters. There has been some recent work on VM workloads characterization [53, 115], but mainly in the public cloud setting. Prior work on measurements of enterprise clusters [43] do not quantify issues related to VM resource allocations. Other measurement studies mainly concentrate on network traffic and communication patterns within data center networks to reduce bandwidth utilization but do not focus per se on VM workload characterization [31].

Profiling RM Approaches: The prior work on resource management based on profiling approaches has focused on empirically deriving application demands by online and offline profiles of real workloads [208, 234, 94].

PseudoApp [202] chooses the right VM size by creating a pseudo application to mimic the resource consumption of a real application; that is, it runs the same set of distributed components and executes the same sequence of system calls as those of the real application. CherryPick [9] leverages Bayesian Optimization to build performance models for various applications, and uses those models to identify the best (or close-to-the-best) configuration, but using extra profiling runs.

Model-Driven RM Approaches: In general, model-driven approaches focus on building models to estimate the impact of different resource allocation strategies on the application performance. Oftentimes, they rely on historical resource demands to train statistical learning models to drive the allocation decisions [195, 235, 84, 186].

PARIS [228] leverages established machine learning techniques, such as random forests, to identify the best VM across multiple cloud providers. Ernest [213] is a system to efficiently run applications on shared infrastructure by choosing the right hardware configuration. Their insight is that a number of jobs have predictable structure in terms of computation and communication, thus they build performance models that can predict the running time (or other performance metric of interest) of jobs on specified hardware configurations. One key difference with our approach is that they do not adjust VM resources on-the-fly, rather, their work assumes fix-sized instance types (as is the case of the public cloud), and they aim to choose the optimal instance type (and optimal number of instances) to run a particular job.

Gmach et al. [90] propose a resource allocation system for data center applications that depends on predicting their behavior a priori based on the repetitive nature of their workloads. On a similar note, DejaVu [210] identifies a few workload categories and leverages them to reuse previous resource allocations so as to minimize re-allocation overheads. In contrast, we assume our workload patterns can change over time, thus we propose a contextual bandits model to dynamically adapt to changes.

Soror et al. [192] leverages cost models that are built into database query optimizers to recommend workload-specific VM configurations. However, their framework only works for SQL-like workloads, as opposed to ours, which is agnostic to the application. Finally, PRESS [92] extracts dynamic patterns in application resource demands and adjusts their resource allocations automatically using signal processing and statistical learning algorithms. They only tune VM CPU limits, although they mention their approach is extensible to other resources, such as memory and networking.

Adaptive RM Approaches: Some prior work investigate auto-scaling using adaptive control loops and reinforcement learning [30, 70, 69, 79, 236, 111, 160, 171, 37], though none of the above use the contextual bandits framework. Other adaptive auto-scaling systems, such as the ones offered in Google Cloud Platform [93] or Amazon Web Services [10], allow users to maintain application availability by dynamically scaling their resources according to conditions they define. For example, users can set target utilization metrics (e.g., average CPU utilization, requests per second) and the system will then automatically adjust the number of instances as needed to maintain those targets (similar to *reactive*). Such systems mainly focus on horizontal scaling, whereas our work targets vertical one. In general, these threshold-based systems (either horizontal/vertical) are simple to implement and use, however their performance depends on the quality of the thresholds [14].

Perhaps the most prominent work on the VM resource allocation problem has been done by Delimitrou et al. [66, 64]. They mainly use collaborative filtering techniques to classify workloads using four different classification tasks (scale up/out, heterogeneity, and interference), and they rely on (small) online workload profiling as well as monitoring tasks for allocation re-adjustment.

Scheduling/Migration Approaches A great deal of previous research into resource management has focused on VM/task scheduling and migration [158, 29, 229, 65, 177]. They are somewhat orthogonal to our work, as we focus on the problem of maximizing the resource usage efficiency of VMs, which should result in easier scheduling, i.e., packing of smaller VMs [98].

4.5 Discussion and Future Work

Although we have proposed an initial framework for adjusting vCPUs and memory of VMs on-the-fly using ML techniques, some natural extensions of this work come to mind, both from a systems as well as an ML perspective. On the systems front, besides improving our simulator and adding support for more application-level metrics (e.g., SQL transactions per second), we are also planning on being able to tune other type of resources, such as networking and storage, as well as managing other entities, such as containers. Further, including sensitivity analyses of the different threshold choices (e.g., 25%, 75%), as well as augmenting the experiments with real workloads would be an obvious step to follow. Regarding ML, apart from experimenting with more complex models, an interesting step to take would be to enable smarter filtering policies in FS. By borrowing ideas from active learning literature [183], we could potentially filter the VMs that would provide the most useful information to our agent. For example, we could pick the instances in a greedy fashion, according to some informativeness measure used to evaluate all the instances in the cluster, or select the most “diverse" instances using submodularity [120], which would allow the agent to have a better coverage of the state space, thus improving generalization and speeding up training.

4.6 Summary

Virtual execution environments enable a more efficient use of server’s resources by consolidating multiple applications onto the same physical hardware. However, provisioning a VM with more (or less) resources than it requires can drastically impact its performance as well as that of other VMs in the cluster.

In this work, we first provided a characterization of resource allocation and utilization of virtual machines from thousands of enterprise clusters running production workloads. Given that we observed a high degree of overprovisioning and underprovisioning, mainly due to inaccurate user guesses, as well as significant variability in load demands over time, we proposed ADARES, an adaptive system that dynamically tunes resources of VMs. ADARES proposes an *ML-based mechanism* that uses the *contextual bandits* framework together with transfer learning to optimize configurations of VMs in a cluster, and exploits cluster, node and VM-level information to promote efficient resource utilization across VMs. Our empirical results showed that our approach can significantly improve system utilization without sacrificing performance.

5 | PULPO

Latency to end-users and regulatory requirements push large companies to build data centers all around the world. The resulting data is "born" geographically distributed. On the other hand, many machine learning applications require a global view of such data in order to achieve the best results. These types of applications form a new class of learning problems, which we call geo-distributed machine learning (GDML). Such applications need to cope with: 1) scarce and expensive cross-data center (X-DC) bandwidth, and 2) growing privacy concerns that are pushing for stricter data sovereignty regulations.

Current solutions to learning from geo-distributed data sources revolve around the idea of first centralizing the data in one data center, and then training locally. As machine learning algorithms are communication-intensive, the cost of centralizing the data is thought to be offset by the lower cost of intra-data center (in-DC) communication during training.

In this work, we show that the current centralized practice can be far from optimal and propose an *ML-System co-design* for enabling geo-distributed training. Herein, we present PULPO, a system that treats *ML as a first-class citizen*, and leverages optimization-based techniques to reduce X-DC center communication. Further, we argue that the geo-distributed approach PULPO enables is structurally more amenable to dealing with regulatory constraints, as raw data never leaves the source data center. Our empirical evaluation on three real datasets shows orders of magnitude improvements in terms of cross-data center bandwidth consumption.

In summary, the main contributions of this work are:

- We introduce GDML, an important class of learning system problems that deals with geo-distributed datasets, and provide a study of the relative merits of state-of-the-art centralized solutions versus geo-distributed alternatives.

- We propose PULPO, a system co-designed with machine learning to enable geo-distributed machine learning. PULPO builds upon Apache Hadoop YARN [211] and Apache REEF [221], and extends their functionality to support multi-data center machine learning applications. We adopt a communication-sparse learning algorithm [139], originally designed to accelerate learning, and leverage it to optimize wide-area bandwidth consumption.

- We present empirical results from both simulations and a real deployment across continents, which show that, under common conditions, geo-distributed approaches can trade manageable penalties in training latency (less than $5\times$) for massive bandwidth reductions (multiple orders of magnitude).

We structure this chapter as follows: Section 5.1 formalizes the problem setting. We then describe the ML-System co-design in Section 5.2 and show evaluation results in Section 5.3. We present related work in Section 5.4, future work in Section 5.5, and summarize in Section 5.6.

5.1 Problem Formulation

In order to facilitate a study of the state-of-the-art centralized approach in contrast to geo-distributed alternatives (Figure 5.1), we formalize the problem in two dimensions: 1) we specify assumptions about the data, its size and partitioning, and 2) we restrict the set of learning problems to the well known statistical query model class [114].

5.1.1 Data distribution

We assume the dataset D of N examples (x_i, y_i), where $x_i \in \mathbb{R}^d$ denotes the feature vector and $y_i \in \{-1, 1\}$ denotes the label of example i, to exist in $p \in \{1, \ldots, P\}$ partitions D_p, each of which is

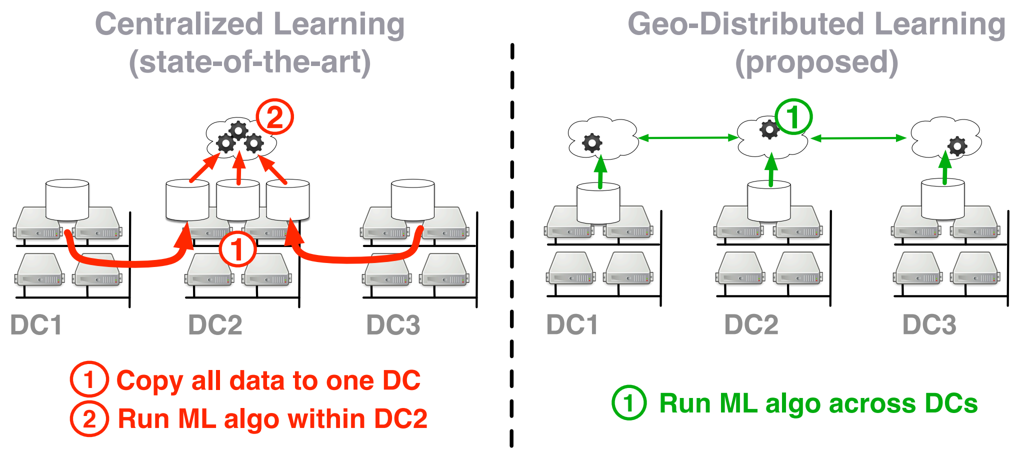


Figure 5.1: Centralized vs. Geo-distributed Learning

generated in one of P data centers. Those P partitions consist of n_p examples each, with $N = \sum_p n_p$. Let d be the dimensionality of the feature vectors and $\bar{d}$ the average sparsity (number of non-zeros) per example. The total size of each partition can be (roughly) estimated as $s_p = n_p\,\bar{d}$. Although this approximation serves our purposes, in order to have a more precise estimate of the partition sizes, we should not rely on a single $\bar{d}$ value (average instance sparsity across the entire data). This is because the sparsity of instance vectors may depend on the data center location. For example, in the case of a recommendation application, the US data center might have more dense feature vectors (user profiles) than those in a data center in South America.

Further, let p^* be the index of the largest partition, i.e., $p^* = \arg\max_p n_p$. Then, the total X-DC transfer needed to centralize the dataset is:

$$T_C = (N - n_{p^*})\,\bar{d} \tag{5.1}$$

The goal here is to transfer all instances to the data center that holds the largest subset of instances. Data compression is commonly applied to reduce this size, but only by a constant factor.

5.1.2 Learning Task

Herein, we restrict the set of learning problems we consider to those that fit the statistical query model. This model covers a wide variety of important practical machine learning models, including both supervised (e.g., linear and logistic regression, support vector machines) and unsupervised (e.g., k-means clustering) models.

The *beauty* of algorithms that fit into the statistical query model is that they can be written in certain summation form, which allows them to be easily parallelized [50]. In the statistical query model, the learning algorithm is allowed to obtain estimates of statistical properties of the examples (e.g., sufficient statistics, gradients) but cannot see the examples themselves [78]. In other words, the algorithm can be phrased purely in terms of statistical queries over the data, and those statistical queries decompose into the sum of a function applied to each example; a structural property we can leverage to co-design our system with ML.

Let that function be denoted by $f_q \in \{f_1, f_2, \dots f_Q\}$. A query result F_q is then computed as $F_q = \sum_{i=1}^{N} f_q(x_i, y_i)$. With the data partitioning, this can be rephrased as

$$F_q = \sum_{p=1}^{P} \sum_{i=1}^{n_p} f_q(x_i, y_i) \tag{5.2}$$

In other words, the X-DC transfer per statistical query is the size of the output of its query function f_q, which we denote as s_q. The total X-DC transfer then depends on the queries and the number of such queries issued, n_q, as part of the learning task, both of which depend on the algorithm executed and the dataset. Let us assume we know these for a given algorithm and dataset combination. Then, we can estimate the total X-DC transfer cost of a fully distributed execution as:

$$T_D = (P-1) \sum_{q=1}^{Q} n_q \, s_q \tag{5.3}$$

Note that this relies on the cumulative and associative properties of the query aggregation, i.e., the problem structure, by which we only need to communicate one result of size s_q per data center. The data center that aggregates the results does not need to communicate any data over the wide-area network, thus the $P-1$ term in Equation (5.3).

With this formalization, the current state-of-the-art approach of centralizing the data relies on the assumption that $T_C \ll T_D$. However, it is not obvious why this should always be the case, as the X-DC transfer of the centralized approach T_C grows linearly with the dataset size, whereas the X-DC transfer of a distributed approach T_D grows linearly with the size and number of the queries. Additionally, relatively large partitions per data center typically yield more meaningful statistics per data center. This, in turn, means that the learning algorithm needs *fewer* queries to converge, lowering T_D as the dataset size grows given a fixed number of partitions.

Hence, it is apparent that the assumption that $T_C \ll T_D$ holds for some, but not all regimes. All things being equal, it seems that larger datasets would favor the distributed approach. Similarly, all things being equal, larger query results and algorithms issuing more queries seem to favor the centralized approach.

In order to study this more precisely, we need to restrict the discussion to a concrete learning problem for which the queries q, their functions f_q and their output sizes s_q are known. Further, the number of such queries can be bounded by invoking the convergence theorems for the chosen learning algorithm. Herein, we choose linear modeling to be the learning problem for its simplicity and rich theory. In particular, we consider the l_2 regularized linear learning problem.

Let $l(w\,x_i, y_i)$ be a continuously differentiable loss function with Lipschitz continuous gradient, where $w \in \mathbb{R}^d$ is the weight vector. Let $L_p(w) = \sum_{i \in D_p} l(w\,x_i, y_i)$ be the loss associated with data center p, and $L(w) = \sum_p L_p(w)$ be the total loss over all data centers. Our goal is to find w that minimizes the following objective function, which *decomposes* per data center:

$$f(w) = \frac{\lambda}{2}||w||^2 + L(w) = \frac{\lambda}{2}||w||^2 + \sum_p L_p(w) \tag{5.4}$$

where $\lambda > 0$ is the regularization constant. Depending on the loss chosen, this objective function covers important cases such as linear support vector machines (hinge loss) and logistic regression (logistic loss). Learning such model amounts to optimizing Equation (5.4). Many optimization algorithms are available for the task, and in Section 5.2.3 we describe the one we choose.

It is important to note that one common statistical query of all those algorithms is the computation of the gradient of the loss in Equation (5.4) with respect to w. The size of that gradient (per partition and globally) is d. Hence, s_q for this class of models can be approximated by d. This allows us to rephrase the trade-off mentioned above. All things being equal, datasets with more examples (x_i, y_i) would tend to favor the distributed approach. Similarly, all things being equal, datasets with higher dimensionality d would generally lean towards the centralized setting.

5.2 System Design

This section describes the co-design of PULPO with machine learning to enable efficient geo-distributed training. Our system relies on optimization-based techniques to efficiently communicate dataset statistics through the wide-area network, thus reducing X-DC bandwidth consumption. Herein, we provide a high-level description of the goals that influenced the design of PULPO, the description of its core components, and the optimization procedure we use to reduce the cross-data center transfers.

5.2.1 Goals

PULPO is designed to efficiently train machine learning models from geo-distributed datasets. We design our system considering the following properties:

- *Flexibility:* We need a flexible system that can run in two regimes, i.e., in-DC and X-DC, without requiring two separate implementations. Further, our system should *not be tied* to any specific algorithm, rather, it should expose a generic framework suitable for geo-distributed and centralized implementations of, at least, ML algorithms expressible in the statistical query model class.

- *Visibility:* The system should provide *enough* visibility of the underlying network topology to the application layer, so that algorithm authors' can explicitly control what to run within a data center and what to run across data centers. Perhaps surprising to software engineers but not to algorithm developers, our system should make the geo-distribution *less* transparent by allowing network-aware placement of tasks.

- *Uniformity:* The system should be able to obtain resources (CPU cores and RAM) across different data centers in a uniformly basis. In other words, it should be able to *view* multiple data centers as a single one. Although this goal seems to clash with the visibility one, they are complementary; that is, uniformity refers to resource management whereas visibility focus on computation and communication patterns.

5.2.2 Components

This section provides an overview of our system and introduce its core components. Figure 5.2 shows a high level overview its three-tier architecture as well as the abstractions each layer provides.

5.2.2.1 Resource Management: Apache Hadoop YARN

PULPO's bottom layer consists of a resource manager. A resource manager is a platform that dynamically leases resources, known as containers, to various competing applications in a cluster. It acts as a central authority and negotiates with potentially many application masters the access to those containers [221]. Among the most well-known implementations are Apache Hadoop YARN [211], Apache Mesos [99] and Google Omega [182]. All of these systems are designed to operate *within* one data center and multiplex applications on a collection of shared machines.

In our setting, we need a similar abstraction, but it must *span* multiple data centers. We build our solution on top of Apache Hadoop YARN. As part of Microsoft's effort to scale-out YARN to Microsoft-scale clusters (tens of thousands of nodes), they have been contributing to Apache a new architecture that allows to *federate* multiple clusters [51]. This effort was not originally intended to operate in a X-DC setting, and as such, was focused on *hiding* from the application layer the different sub-clusters. It is worth mentioning that single data center federation is deployed in

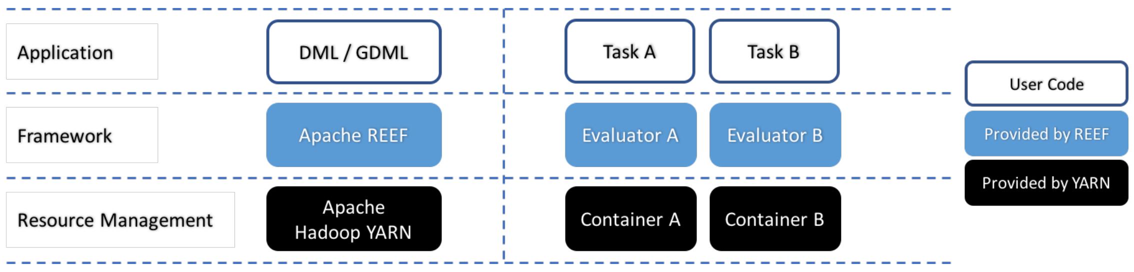


Figure 5.2: PULPO Architecture

production at scale at Microsoft.

As part of this work, we have been experimenting and extending this system, leveraging its transparency yet providing enough visibility of the network topology to our application layer. As a result, we can run a *single* application that spans different data centers in an efficient manner.

5.2.2.2 Framework: Apache REEF

On top of YARN, PULPO uses Apache REEF [221], which provides the basic control flow for our application. This middleware provides a generalized control plane to ease the development of applications on resource managers. REEF provides a control flow master called Driver to applications, and an execution environment for tasks on containers called Evaluator. Applications are expressed as event handlers for the Driver to perform task scheduling (including fault handling) and the task code to be executed in the Evaluators. As part of this work, we extend REEF to support geo-federated YARN, including scheduling of resources to particular data centers.

REEF provides a group communications library that exposes Broadcast and Reduce operators similar to Message Passing Interface (MPI) [96]. Like MPI, REEF's group communications library is designed for the single data center case. We expand it to cover the X-DC case we study here. Note that all changes to Apache REEF have been contributed back to the project where appropriate.

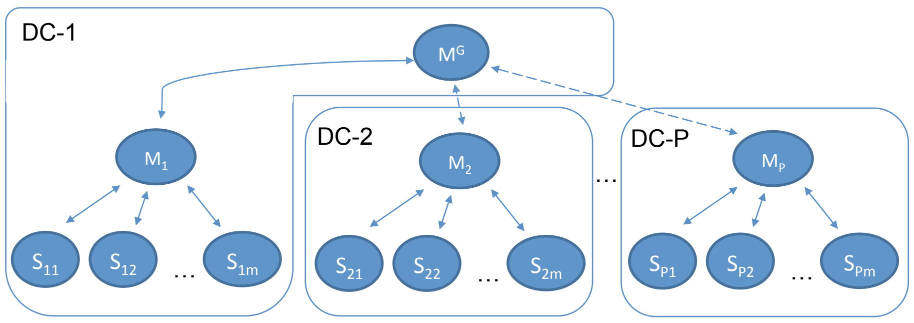


Figure 5.3: Multi-Level Master/Slave Communication Tree with P data centers, each with its own data center master (M_i) and slaves (S_{ij}). The global master M^G is physically located in DC-1. The solid and dashed lines refer to in-DC and X-DC links respectively.

5.2.2.3 Application Layer: DML / GDML

Statistical query model algorithms can be implemented using nothing more than Broadcast and Reduce operators [50], where data partitions reside in each machine, and the statistical query is Broadcast to those, while its result is computed on each machine and then aggregated via Reduce.

Both Broadcast and Reduce operations are usually implemented via communication trees in order to maximize the overall throughput of the operation. Traditional systems, such as MPI [96] implementations, derive much of their efficiency from intelligent (and fast) ways to establish these trees. Different from the in-DC environment where those are typically used, our system needs to work with network links of vastly different characteristics. X-DC links have higher latency than in-DC ones, whereas the latter have usually higher bandwidths [22]. Further, WAN links are much more expensive than intra-data center links, as they are frequently rented or charged-for separately, in the case of the public cloud.

PULPO addresses these challenges with a heterogeneous, multi-level communication tree. Figure 5.3 shows an example of the multi-level communication tree we use. A global Broadcast originates from M^G to the data center masters M_i, which in turn do a local Broadcast to the slave nodes S_{ij} in their own data centers. Conversely, local Reduce operations originate on those slave nodes, while the data center masters M_i aggregate the data prior to sending it to M^G for global

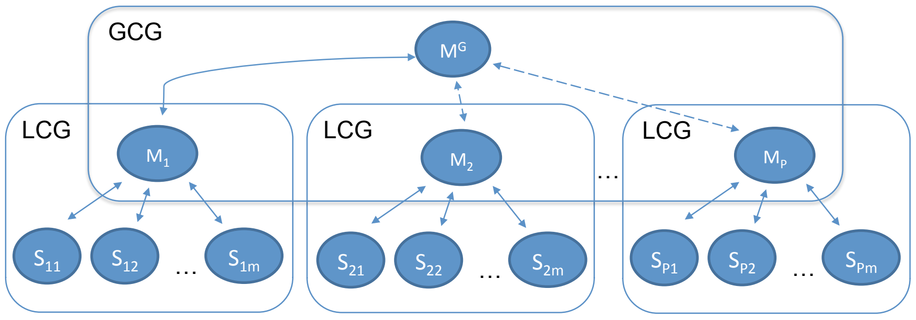


Figure 5.4: Communication Groups view of the Multi-Level Master/Slave Communication Tree depicted in Figure 5.3. There are P local communication groups (LCG) that connect the data center masters (M_i) with their respective slaves (S_{ij}) to enable local Broadcast / Reduce operations. Further, a single global communication group (GCG) enables the interaction between the global master M^G and the data center masters M_i for global aggregation.

aggregation.

To make this happen, we extend REEF's communication library in order to create multiple communication groups, as shown in Figure 5.4. The global master M^G together with the data centers masters M_i, form the global communication group (GCG), where the global Broadcast / Reduce operations are performed, and used in the outer loop of Algorithm 1—as we shall see next. Likewise, the slave nodes within each data center and their masters M_i form the local communication groups (LCG), where the local Broadcast / Reduce operations execute, and are used to optimize the local approximations $\hat{f}_p$ of Equation (5.8) (more details next). This mechanism provides the flexibility to communicate only locally (LCG) or globally between masters (GCG), and highlights the *ML as a first-class citizen* property our system embraces.

5.2.3 Optimization-based Approach

Given the iterative nature of ML optimization algorithms, we need to somehow reduce the number of X-DC iterations if we want our geo-distributed training approach to reduce the wide-area communication costs when compared to centralized methods. In other words, we need a

Algorithm 1 Functional Approximation based Distributed Learning Algorithm (FADL)

```
Choose $w^0$
for $r = 0, 1...$ do
    Compute $g^r$ (X-DC communication)
    Exit if $||g^r|| \leq \varepsilon_g ||g^0||$
    for $p = 1, ..., P$ (in parallel) do
        Construct $\hat{f}_p(w)$ (Equation (5.8))
        $w_p \leftarrow$ Optimize $\hat{f}_p(w)$ (in-DC communication)
    end for
    $d^r \leftarrow \frac{1}{P}\sum_p w_p - w^r$ (X-DC communication)
    Line Search to find $t$ (negligible X-DC communication)
    $w^{r+1} \leftarrow w^r + t\, d^r$
end for
```

communication-efficient algorithm capable of minimizing the communication between the data centers. It is clear from Equation (5.3) that such an algorithm should try to minimize the number of queries whose output size is very large. In the case of Equation (5.4), this means that the number of X-DC gradient computations should be reduced.

Recently, many communication-efficient algorithms have been proposed that trade-off local computation with communication [5, 114, 35, 232, 105]. Herein, we use the algorithm proposed by Mahajan et al. [140] to optimize Equation (5.4), shown in Algorithm 1. We choose this algorithm because experiments show that it performs better than the aforementioned ones, both in terms of communication and running time [140]. The algorithm was initially designed for running in a traditional distributed machine learning setting, i.e., single data center.[1] In this work, we adapt it for X-DC training, a novel application that was not originally intended for.

The main idea of the algorithm is to trade-off in-DC computation and communication with X-DC communication. The minimization of the objective function $f(w)$ is performed using an

[1]We confirmed this with the first author.

iterative descent method in which the r^{th} iteration starts from a point w^r, computes a direction d^r, and then performs a line search along that direction to find the next point $w^{r+1} = w^r + t\, d^r$.

We adapt the algorithm to support GDML in the following manner. Each node in the algorithm now becomes a data center. All the local computations like gradients and loss function on local data now involves both computation as well as in-DC communication among the nodes in the same data center. On the other hand, all global computations like gradient aggregation involves X-DC communication. This strengthens the need for the two levels of communication and control described in the previous section, and reinforces the choice of an *ML-System co-design* to enable efficient geo-distributed machine learning.

In a departure from communication-heavy methods, this algorithm uses distributed computation for generating a good search direction d^r in addition to the gradient g^r. At iteration r, each data center has the current global weight vector w^r and the gradient g^r. Using its local data D_p, each data center can form an approximation $\hat{f}_p$ of f. To ensure convergence, $\hat{f}_p$ should satisfy a gradient consistency condition, $\nabla \hat{f}_p(w^r) = g^r$. The function $\hat{f}_p$ is approximately[2] optimized using a method M to get the local weight vector w_p, which enables the computation of the local direction $d_p = w_p - w^r$. The global update direction is chosen to be $d^r = \frac{1}{P} \sum_p d_p$, followed by a line search to find w^{r+1}.

In each iteration, the computation of the gradient g^r and the direction d^r requires communication across data centers. Since each data center has the global approximate view of the full objective function, the number of iterations required are significantly less than traditional methods, resulting in orders of magnitude improvements in terms of X-DC communication.

The algorithm offers great flexibility in choosing $\hat{f}_p$ and the method M used to optimize it. A general form of $\hat{f}_p$ for Equation (5.4) is given by:

$$\hat{f}_p(w) = \frac{\lambda}{2} ||w||^2 + \tilde{L}_p(w) + \hat{L}_p(w) \tag{5.5}$$

where $\tilde{L}_p$ is an approximation of the total loss L_p associated with data center p, and $\hat{L}_p(w)$ is

[2]Mahajan et al. [140] proved linear convergence of the algorithm even when the local problems are optimized approximately.

an approximation of the loss across all data centers except p, i.e., $L(w) - L_p(w) = \sum_{q \neq p} L_q(w)$. One can simply use $\tilde{L}_p = L_p$, i.e., the exact loss function for data center p. However, Mahajan et al. [140] showed better results if the local loss function is also approximated. Among the possible choices suggested, we consider the following quadratic approximations in our work:

$$\tilde{L}_p(w) = \nabla L_p(w^r)(w - w^r) + \frac{1}{2}(w - w^r)^T H_p^r (w - w^r) \tag{5.6}$$

$$\hat{L}_p(w) = (\nabla L(w^r) - \nabla L_p(w^r))(w - w^r) + \frac{P-1}{2}(w - w^r)^T H_p^r (w - w^r) \tag{5.7}$$

where H_p^r is the Hessian of L_p at w^r. Replacing (5.6) + (5.7) in (5.5) we have the following objective function:

$$\hat{f}_p(w) = \frac{\lambda}{2}||w||^2 + g^r(w - w^r) + \frac{P}{2}(w - w^r)^T H_p^r (w - w^r) \tag{5.8}$$

We use the conjugate gradient (CG) algorithm [185] to optimize (5.8). Note that each iteration of CG involves a statistical query with output size d to do a hessian-vector computation. However, this query involves only in-DC communication and computation, whereas for traditional second order methods like TRON [136], it will involve X-DC communication.

Discussion Let T_{outer} be the number of iterations required by the algorithm to converge. Each iteration requires two queries with output size $s_q = d$ for the gradient and direction computation, and few queries of output size $s_q = 1$ for the objective function computation in the line search. Since $d \gg 1$, we can ignore the X-DC communication cost for the objective function computation. Hence, we can rewrite Equation (5.3) as $T_D = 2\,(P-1)\,d\,T_{outer}$. Therefore, for T_D to be less than T_C the following must hold:

$$2\,(P-1)\,d\,T_{outer} \quad < \quad (N - n_{p^*})\,\bar{d} \tag{5.9}$$

In practice, the typical value of P (data centers) is relatively small (in the 10s). Since there are few data centers (i.e., few partitions of the data), the above algorithm will take only few (5-20) outer iterations to converge. In fact, in all our experiments in Section 5.3, the algorithm converges in less than 7 iterations. This means that as long as the total size of the data is roughly more than $2-3$ orders of magnitude greater than the dimensionality d, which is the typical case for large datasets, doing *optimized* geo-distributed learning would reduce the X-DC transfers compared to the centralized approach.

5.3 Evaluation

We implemented a prototype of PULPO in about 13 kLOC of Java, which allows us to evaluate the state-of-the-art of centralizing the data before learning in comparison with truly distributed approaches. In this section, we describe our findings, starting with the setup and definition of the different methods used, followed by results from both simulated and real deployments.

5.3.1 Setup

Clusters We report experiments on two deployments: a distributed deployment on Azure across two data centers, and a large centralized cluster on which we simulate a multi-data center setup (2, 4, and 8 data centers). This simulated environment is our main testbed, and we mainly use it for multi-terabyte scale experiments, which are not cost-effective on public clouds. We use 256 slave nodes divided into 2-8 simulated data centers in all our simulations. Further, all the experiments are done with the logistic loss function.

We ground and validate the findings from the simulations on a real cross-continental deployment on Microsoft Azure. We establish two clusters, one in a data center in Europe and the other on the U.S. west coast. We deploy two DS12 VMs into each of these clusters. Each of those VMs has 4 CPU cores and 28 GB of RAM. We establish the site-to-site connectivity through a VPN tunnel using a High Performance VPN Gateway.[3]

Datasets We use three datasets of user behavior data in web sites for our evaluation, two of which are publicly available. All of them are derived from click logs. Table 5.1 summarizes their

[3]More details in https://azure.microsoft.com/en-us/documentation/articles/vpn-gateway-about-vpngateways/.

Dataset	Examples	Features	Size	
	(N)	(d)	Model	Dataset
CRITEO	4B	5M	20MB	1.5TB
		10M	40MB	1.5TB
		50M	200MB	1.6TB
		100M	400MB	1.7TB
WBCTR	730M	8M	32MB	347GB
		16M	64MB	362GB
		80M	320MB	364GB
		160M	640MB	472GB
KAGGLE	46M	0.5M	2MB	8.5GB
		1M	4MB	8.5GB
		5M	20MB	9GB

Table 5.1: Datasets overview. Dataset sizes reported are *after* compression. Weights in the models are represented in single-precision floating-point format (32 bits) with no further compression. Note that the average sparsity (number of non-zeros) of each of the dataset versions are very similar, thus we do not observe a significant size change after increasing the number of features.

statistics. CRITEO and KAGGLE are publicly available [56, 110]. The latter is a small subset of the former, and we use it for the smaller scale experiments in Azure. WBCTR is an internal Microsoft dataset. We vary the number of features in our experiments using hashing kernels as suggested in [222].

The dataset sizes reported in Table 5.1 refer to compressed data. The compression and decompression is done using Snappy,[4] which enables high-speed compression and decompression with

[4]For more details refer to https://github.com/xerial/snappy-java.

reasonable compression size. In particular, we achieve compression ratios of around 62-65% for the CRITEO and WBCTR datasets, and 50% for KAGGLE. Our system performs all computations using double precision arithmetic, but communicates single precision floats. Hence, model sizes in Table 5.1 are reported based on single-precision floating point numbers.

In our experiments, we assume the datasets are randomly partitioned across the data centers, i.e., we assume each data center keeps an approximately equal number of examples. Note that although this is a strong assumption, as data in different data centers can be distributed differently, it holds true in some important production use cases we observe. In such cases, load balancing across data centers forces data to be "randomly" spread across them. However, this is not fully general, as other important GDML workloads require data to be close to the users (to achieve low latency interactions), thus strong geographically biases emerge. Besides the dataset sizes, in practice, data centers can also vary significantly in terms of their bandwidth and computational resources. We plan on addressing these issues in future work.

Methods We contrast the state-of-the-art approach of centralizing the data prior to learning with several alternatives, both within the regime requiring data copies and truly distributed:

- *centralized*, denotes the current state-of-the art, where we copy the data to one data center prior to training. Based on the data shipping model used, two variants of this approach arise:
  - *centralized-stream*, refers to a streaming copy model where the data is replicated as it arrives. When the learning job is triggered in a particular data center, the data has already been transferred there, therefore, no copy time is included in the job running time, and
  - *centralized-bulk*, refers to a batch replication scheme where the data still needs to be copied by the time the learning process starts, therefore, the copy time has an impact on the job running time, i.e., the job needs to wait until the transfer is made to begin the optimization.

- *distributed*, which builds the multi-level master/slave tree for X-DC learning, but does not use the efficient algorithm in Section 5.2.3 to optimize Equation (5.4), instead, it optimizes using TRON [136].

- *distributed-fadl*, which uses the algorithm introduced in Section 5.2.3 to optimize Equation (5.4), and similarly to *distributed*, it performs the optimization in a geo-distributed fashion, i.e., it leaves the data in place and runs a single job that spans training across data centers.

We observe both flavors of centralized to occur in practice. We simply refer to *centralized* when no distinction between its variants is required. This approach (and its variants) only performs *compressed* data transfers, and uses the algorithm described in Section 5.2.3 for solving the l_2 regularized linear classification problem mentioned in Section 5.1.

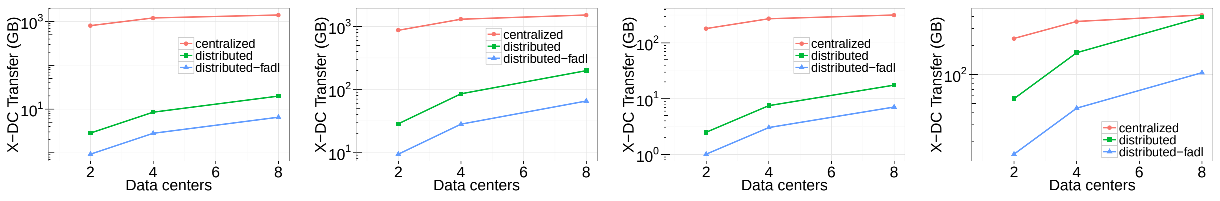
Both *distributed* and *distributed-fadl* methods represent the furthest departure from the current state-of-the-art as their execution is truly geo-distributed. Studying results from both allows us to draw conclusions about the relative merits of the system enabling truly geo-distributed training (*distributed*) as well as the optimization-based technique used to minimize the system's X-DC bandwidth consumption (*distributed-fadl*).

5.3.2 Results

In this section we present results from the methods introduced above. We focus on two key metrics: 1) total X-DC transfer size, and 2) latency to model.

5.3.2.1 Simulation

X-DC Transfer Figure 5.5 illustrates the total X-DC transfer of the different methods for different numbers of data centers. We only show two versions of CRITEO and WBCTR, though the others follow the same patterns. In general, X-DC transfers increase with the number of data centers as there are more wide-area communication paths. As expected, increasing the model dimensionality also impacts the transfers in the distributed versions. In Figure 5.5b, the optimized

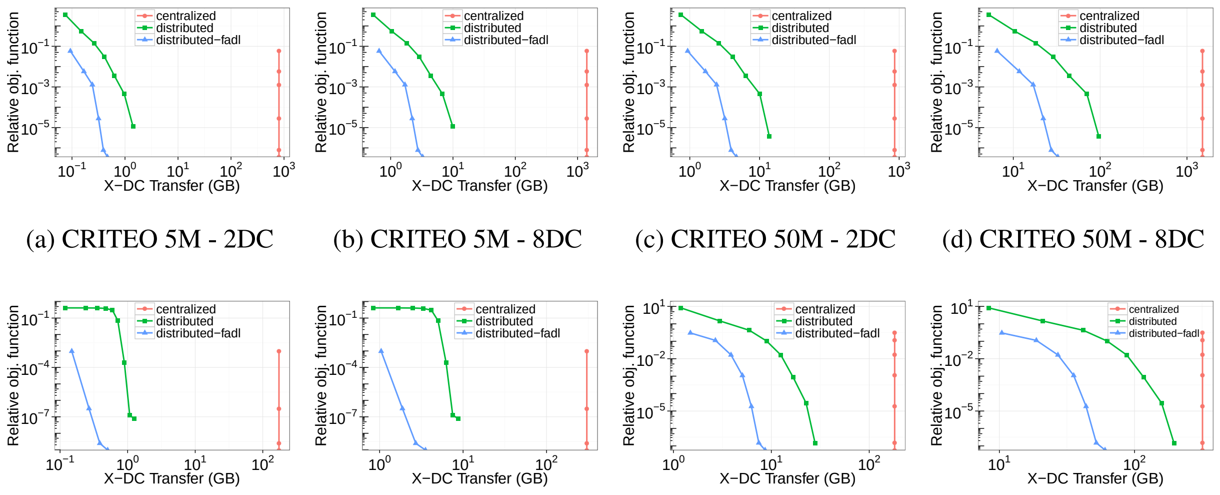


(a) CRITEO 10M (b) CRITEO 100M (c) WBCTR 16M (d) WBCTR 160M

Figure 5.5: X-DC transfer (in GB) versus number of data centers for two versions of CRITEO and WBCTR datasets (y-axis is in log scale). The method *distributed-fadl* consumes orders of magnitude less X-DC bandwidth than any variant (*stream* or *bulk*) of the compressed *centralized* approach. Moreover, a naive algorithm that does not economize X-DC communication, as is the case of the *distributed* method, also reduces transfers with respect to the current *centralized* state-of-the-art.

distributed approach (*distributed-fadl*) performs at least one order of magnitude better than *centralized* in every scenario, achieving the biggest difference (2 orders of magnitude) for 2 data centers. In this setting, *centralized* (any variant) transfers half of the compressed data (870 GB) through the X-DC link before training, whereas *distributed-fadl* just needs 9 GBs worth of transfers to train the model. Likewise, in the WBCTR dataset (Figure 5.5d), we see the biggest difference in the 2 data centers scenario (1 order of magnitude). When the data is spread across 8 data centers, *centralized* transfers almost the same as *distributed*. In general, even the non communication-efficient *distributed* baseline also outperforms the current practice, *centralized*, on both datasets.

Objective / X-DC Transfer Trade-off Commercial deployments of machine learning systems impose deadlines and resource boundaries on the training process. This can make it impossible to run the algorithm till convergence. Hence, it is interesting to study the performance of the centralized and distributed approaches in relationship to their resource consumption. Figure 5.6 shows the relative objective function over time as a function of X-DC transfers for 2 and 8 data centers on the CRITEO and WBCTR datasets. We use the relative difference to the optimal function value, cal-



(a) CRITEO 5M - 2DC (b) CRITEO 5M - 8DC (c) CRITEO 50M - 2DC (d) CRITEO 50M - 8DC

(e) WBCTR 8M - 2DC (f) WBCTR 8M - 8DC (g) WBCTR 80M - 2DC (h) WBCTR 80M - 8DC

Figure 5.6: Relative objective function (compared to the best) versus X-DC transfer (in GB) for 2 and 8 data centers for two versions of CRITEO and WBCTR datasets (both axis are in log scale). The method *distributed-fadl* achieves lower objective values much sooner in terms of X-DC transfers than the other methods. The *centralized* objective remains constant with respect to X-DC transfers throughout the optimization as it starts once the data has been transferred. The *distributed* method does incur in more transfers than *distributed-fadl*, although it also reduces the overhead of the *centralized* approach. Increasing the models dimensionality, naturally increases the X-DC transfers. Note that *centralized* refers to both of its variants (*stream* and *bulk*), and we only report compressed data transfers for this method.

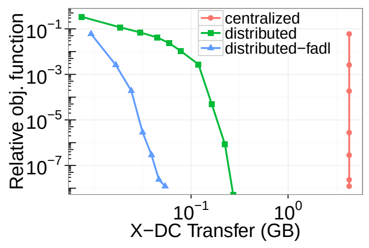
culated as $(f - f^*)/f^*$, where f^* is the minimum value obtained across methods. X-DC transfers remain constant in the *centralized* (any variant) method as it starts the optimization after the data is copied, i.e., no X-DC transfers are made while training. In general, *distributed-fadl* achieves lower objective values much sooner in terms of X-DC transfers, which means that this method can get some meaningful results faster. If an accurate model is not needed (e.g., 10^{-2} relative objective function value), *distributed-fadl* gives a quicker response. As we increase the number of data centers, X-DC communication naturally increases, which explains the right shift trend in the plots (e.g., Figures 5.6a and 5.6b).

Storage As the number of data centers increases, *centralized* (any variant) requires more space on disk. In particular, assuming the data is randomly partitioned across data centers, *centralized* stores at least 1.5× more data than the distributed versions, with a maximum difference of almost 2× when considering 8 data centers. On the other hand, both *distributed* and *distributed-fadl* only need to store the original dataset (1×) throughout the different configurations.

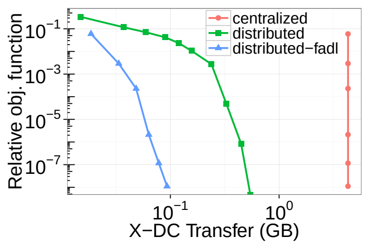
5.3.2.2 Real Deployment

X-DC Transfer To validate our simulation, we include Figure 5.7, which shows the relative objective function with respect to the X-DC bandwidth for the KAGGLE dataset in 2 Azure data centers (Western US and Western Europe). These experiments match our findings in the simulated environment. For the *centralized* approach, we transfer the data from EU to US, and run the optimization in the latter data center. Similar to Figure 5.6, the increase in the number of features causes more X-DC transfers (right shift trend in the plots). The efficient geo-distributed method *distributed-fadl* still communicates the least amount of data, almost 2 orders of magnitude less than the *centralized* (any variant) approach for the 500k model (Figure 5.7a).

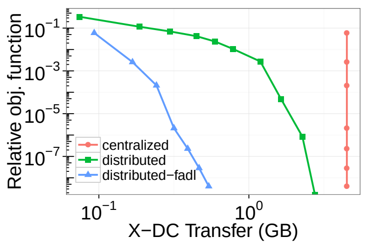
Runtime Figure 5.8 shows the relative objective function over time for the 2 Azure data centers using the KAGGLE dataset. We normalize the time to the *centralized-stream* approach, calculated as t/t^*, where t^* is the overall time taken by *centralized-stream*. This method performs the fastest



(a) KAGGLE 500k



(b) KAGGLE 1M



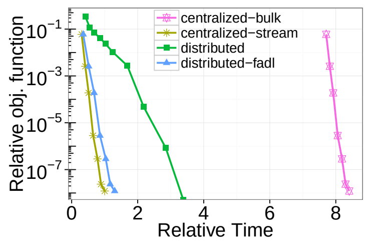
(c) KAGGLE 5M

Figure 5.7: Relative objective function (compared to the best) versus X-DC transfer (in GB) for the KAGGLE dataset in 2 Azure data centers (both axis are in log scale). The increase in the model size explains the right shift trend in the plots. The method *distributed-fadl* consumes the least amount of X-DC bandwidth, at least 1 order less in every scenario, and 2 when using the 500k model. The objective/transfer pattern is similar to Figure 5.6. Both distributed methods transfer much less X-DC data than the *centralized* state-of-the-art. Note that *centralized* refers to both of its flavors (*stream* and *bulk*), and only transfers compressed data.

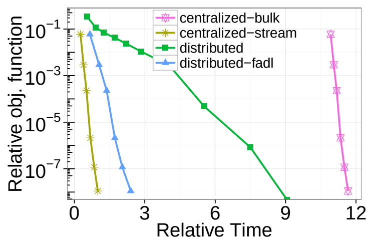
in every version of the dataset (500k, 1M, and 5M features) as the data has already been copied by the time it starts, i.e., no copy time overhead is added, and represents the lower bound in terms of running time.

Although the *centralized* approach always transfers compressed data, we do not take into account the compression/decompression time for computing the *centralized-bulk* runtime, which would have otherwise tied the results to the choice of the compression library. Figures 5.8a, 5.8b, and 5.8c show that *centralized-bulk* pays a high penalty for copying the data, it runs in approximately 8$\times$ or more of its *stream* counterpart.

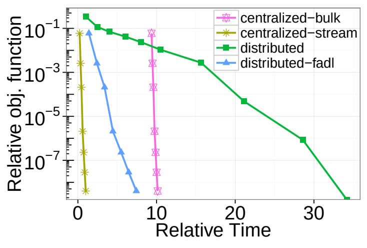
The communication-efficient *distributed-fadl* approach executes in 1.3$\times$, 2.4$\times$, and 7.4$\times$ of the *centralized-stream* baseline for 500k, 1M, and 5M models respectively, which is a remarkable result given that it transfers orders of magnitude less data (Figure 5.7), and executes in a truly geo-distributed manner, respecting potentially strict regulatory constraints. Moreover, if we consider the relative objective function values commonly used in practice to achieve accurate models (10^{-4},



(a) KAGGLE 500k



(b) KAGGLE 1M



(c) KAGGLE 5M

Figure 5.8: Relative objective function (compared to the best) over time (relative to the *centralized-stream* method) for the KAGGLE dataset in 2 Azure data centers (y-axis is in log scale). The method *distributed-fadl* beats every approach but *centralized-stream*. This latter method is the best case scenario, where the data has already been copied and is available in a single data center when the job is executed. The *distributed-fadl* method lies very close to the optimum (*centralized-stream*), especially in low-dimensional models and when considering commonly accepted objective function values (10^{-4}, 10^{-5}). Both *distributed* and *distributed-fadl* performance degrades when the model size increases (as expected), but *distributed* does so much worse (5.8c), which further shows that in order to do geo-distributed machine learning, a communication-efficient algorithm is needed.

10^{-5}), this method's convergence time lies in the same ballpark as the lower bound *centralized-stream* in terms of running time. Still, *distributed-fadl* is way ahead in terms of X-DC transfers (orders of magnitude of savings in X-DC bandwidth), while at the same time it potentially complies with data sovereignty regulations.

Figure 5.8a shows that *distributed-fadl* performs very close to the best scenario, matching the intuition built in Section 5.1 that this method does very well on tasks with (relatively) small models and (relatively) large number of examples. Furthermore, this efficient method also runs faster than *distributed* in every setting, which further highlights the importance and benefits of the ML-System co-design introduced in Section 5.2.

Finally, *distributed* performance degrades considerably as the model size increases. In particular, this method does a poor job when running with 5M features (Figure 5.8c), which concurs with the intuition behind the state-of-the-art *centralized* approach: copying the data offsets the communication-intensive nature of (naive) machine learning algorithms. We see that this intuition does not hold true for the efficient algorithm described in Section 5.2.3.

5.4 Related Work

We discuss work relevant to PULPO in the areas of distributed systems and ML algorithms that deal with disperse datasets.

Distributed Systems Prior work on systems that deal with geographically distributed datasets exists in the literature. The work done by Vulimiri et al. [219, 217] poses the thesis that increasing global data and scarce X-DC bandwidth, coupled with regulatory concerns, will derail large companies from executing centralized analytics processes. They propose a system that supports SQL queries for doing X-DC analytics. Unlike our work, they do not target iterative machine learning workflows, neither do they focus on jobs latency. They mainly discuss reducing WAN data transfer volume.

Pu et al. [167] proposes a low-latency distributed analytics system called Iridium. Similar to Vulimiri et al., they focus on pure data analytics and not on machine learning tasks. Another key difference is that Iridium optimizes task and data placement across sites to minimize query response time, while our system respects stricter sovereignty constraints and does not move raw data around.

JetStream [169] is a system for wide-area streaming data analysis that performs efficient queries on data stored "near the edge". They provide different approximation techniques to reduce the data size transfers at the expense of accuracy. One of such techniques is dropping some fraction of the data via sampling. Similar to our system, they only move *important* data to a centralized location for global aggregation (in our case, we only move statistics and models), and they compute local aggregations per site prior to sending (in our case, we perform local optimizations per data center using the algorithm described in Section 5.2.3). Another streaming application is distributed monitoring, which has focused on continuous tracking of complex queries over collections of physically distributed data streams. Effective solutions in their setting also need to guarantee communication efficiency over the underlying network [128].

Another line of research has focused on multi-site distributed search engines [20, 81]. Such work has shown to reduce the resource consumption in query processing as well as user perceived latency when compared to single-site centralized search engines [41, 113]. It bears some resemblance to our work but in the context of information retrieval.

Other existing Big Data processing systems, such as Parameter Server, Graphlab, or Spark [132, 137, 133, 231], efficiently process data in the context of a single data center, which typically employs a high-bandwidth, relatively low-cost network. To the best of our knowledge, they have not been designed for multi-data center deployments, where scarce WAN bandwidth makes it impractical to naively communicate parameters between locations. Instead, our system was specifically co-designed with ML to perform well on this X-DC setting.

Since our initial work on GDML systems [46, 45], other studies have emerged in the area. Among the most prominent ones we find Gaia [101], which also focuses on leveraging intelligent communication mechanisms, with more emphasis on reducing training times rather than X-DC transfers. Further, the work by Konečný et al. [116, 118, 117] introduces the concept of Federated Learning, where the idea is to train a global model with data residing in mobile devices, instead of data centers. Their setting is very similar to GDML in the sense that communication efficiency is of utmost importance, but the cardinality is quite different (millions of devices as opposed to tens of data centers). This poses other research questions, e.g., what sample of devices to choose at a given point in time, how to alleviate the fact that mobile devices are frequently offline, etc.

Distributed ML Algorithms Besides the systems solutions, the design of efficient distributed machine learning algorithms has also been the topic of a broad research agenda [190, 5, 105, 35, 232, 140, 233, 21, 25, 144]. The general principle has been to trade-off computation and communication, i.e., increase computation in the worker nodes by executing more advanced calculations between each communication round in order to reduce the number of such rounds.

Some recent work uses model quantization, i.e., reduce the number of bits of the model parameters at the expense of potentially losing some accuracy, to reduce the communication cost [197]. Further, a significant fraction of the current research on distributed machine learning pays particular attention to lowering computational costs by using GPUs [48, 4].

The Terascale method [5] might be the best representative method from the statistical query model class and is considered a state-of-the-art solver. CoCoA [191, 105, 189] represents the class of distributed dual methods that, in each outer iteration, solve (in parallel) several local dual optimization problems. Alternating Direction Method of Multipliers (ADMM) [35, 232] is a dual method different from the primal method we use here, however, it also solves approximate problems in the nodes and iteratively reaches the full batch solution. Follow up work [140] shows that the algorithm described in Section 5.2.3 performs better than the aforementioned ones, both in terms of communication rounds and running time.

5.5 Discussion and Future Work

GDML is an interesting, challenging and open area of research. Although we have proposed an initial and novel geo-distributed approach that shows substantial gains over the centralized state-of-the-art in many practical settings, many open questions remain, both from a systems and a machine learning perspective.

Perhaps, the most crucial aspect is fault tolerance. With data centers distributed across continents, consistent network connectivity is harder to ensure than within a single data center, and network partitioning is more likely to occur. On the other hand, a data center level failure might completely compromise the centralized approach (if the primary data center is down), while the geo-distributed solution might continue to operate on the remaining data partitions. There has been some initial work [157] to make ML algorithms tolerant to missing data (e.g., machine failures). This work assumes randomly distributed data across partitions. Hence, a failure removes an unbiased fraction of the data. In production settings, this is the case when multi-data center deployments are created for load-balancing (e.g., within a region)—we are aware of multiple such scenarios within Microsoft's infrastructure. However, cross-region deployments are often dictated by latency-to-end-user considerations. In such settings, losing a data center means losing a heavily biased portion of the population (e.g., all users residing in Western US). Even without fail-stop failures in data centers, the presence of stragglers tasks might impede progress, as we are currently doing synchronous X-DC updates. The obvious thing to do is do asynchronous updates though it may work only under mild conditions. Coping with faults, and tolerating transient or persistent data unavailability and stragglers, as well as understanding the impact of different data distributions in convergence speeds are still open problems that will likely require both systems and ML contributions.

In this work we have restricted ourselves to linear models with l_2 regularization, and shown results on logistic regression models. It would be interesting to validate similar observations in other regularizers (e.g., l_1). More broadly, studying geo-distributed solutions that can minimize X-DC transfers for other complex learning problems such as trees, deep neural networks, etc., is still an open area of research.

Further, a truly geo-distributed approach surely does no worse than a centralized method when analyzed from regulatory and data sovereignty angles. Questions in this area arise not only at the global scale, where different jurisdictions might not allow raw data sharing, but also at the very small scale, e.g., between data stored in a private cluster and data shared in the cloud. We believe that studying the setup presented here from a privacy-preserving and regulatory-compliance angle will yield important improvements, and potentially inform regulators.

One aspect we did not cover is related to the work-cycle of these global data repositories and its impact in the efficacy of geo-distributed learning. If the data gets crunched by x algorithms once it is gathered into a single data center, including, perhaps, by algorithms that depend on each others inputs and/or encompass interactive workflows, the centralized methodology might be more effective than the geo-distributed one. This latter approach would increase communication by x-fold, whereas the centralized method would not incur in any extra communication. We consider that a more in-depth study of which approach (centralized or geo-distributed) is more adequate for different problem settings is still missing. Even more, we have not yet addressed the issue of whether a hybrid method that combines both centralized and geo-distributed learning could be more suitable under certain circumstances.

5.6 Summary

Large organizations have a planetary footprint with users scattered in all continents. Latency considerations and regulatory requirements motivate them to build data centers all around the world. From this, a new class of learning problems emerge, where global learning tasks need to be performed on data “born” in geographically disparate data centers, which we called geo-distributed machine learning (GDML). To the best of our knowledge, this aspect of machine learning has not been studied in great detail before, despite being faced by practitioners on a daily basis.

In this work, we introduced and formalized this problem, and challenged common assumptions and practices. We then presented PULPO, a *system co-designed with machine learning* that enables efficient geo-distributed training. In particular, PULPO treats *ML as a first-class citizen* by leveraging optimization-based techniques in order to reduce X-DC center communication. Our empirical results showed that our geo-distributed system, combined with communication-parsimonious algorithms, can deliver a substantial reduction in costly and scarce cross data center bandwidth.

6 | Conclusions

Distributed systems consist of many interconnected components that interact with each other to perform certain task(s). Many of these systems typically rely on heuristics or sets of rules to make decisions, as well as carefully-engineered analytical models. However, their design is nontrivial.

The same system may need to work under widely different conditions, and handle heterogeneous workloads that might be non-stationary and change over time. All these, together with the intrinsic complexity driven by the large number of components, and the irregular interactions and resource needs, make it difficult to reason about distributed systems' performance in general.

In this thesis, we proposed *optimizing distributed systems using machine learning* techniques, in order to bridge the gap of systems performance and make them more efficient and responsive to varying operating conditions. Depending on the control we have on the system itself and the characteristics and constraints of the problem domain, we identified three main modeling strategies.

First, in *black-box* distributed systems, where we do not have much control over the internals but instead we can only scratch the surface and alter some of the decision-making processes, machine learning can be used as an enabler for new system policies: *ML-based policies*. In such a role, ML would not be as tightly integrated to the underlying system itself, therefore, it may face situations where it is not aware of how the system is doing and receive delayed performance feedback. In those cases, we proposed using *reinforcement learning-based* techniques.

Second, in *gray-box* distributed systems, where we have somewhat more control and can alter their internals, machine learning can be used to provide new system mechanisms: *ML-based mechanisms*. This puts ML in a more powerful position within the system, which may allow it to have fine-grained understanding of what is happening and receive immediate performance indicators. In such cases, we justified the use of *bandit-based* modeling approaches.

Third, in *white-box* distributed systems, where we have full control, we can empower machine learning to play an even more fundamental role: treat *ML as a first-class citizen*. Herein, we proposed an *ML-System co-design* to enable ML-aware systems and systems-aware ML in order to promote a new generation of intrinsically smart, self-tuning, and self-adaptive systems.

The main contribution of this thesis was the design, implementation, augmentation, and evaluation of three distributed systems that illustrated the impact of these machine learning-based optimizations: 1) CURATOR, a framework that safeguards distributed storage systems' health and performance leveraging *ML-based policies* to schedule background maintenance tasks using *reinforcement learning*, 2) ADARES, an adaptive system that relies on an *ML-based mechanism* to dynamically adjust virtual machine resources in virtual executing environments using *bandit-based techniques*, and 3) PULPO, a federation-based *system co-designed with machine learning* optimization techniques to efficiently train models across different data centers.

Each system instantiated appropriate ML models for the task at hand, alleviating systems designers from the responsibility of manually tuning rules and handcrafting complex analytical models. Along the way, we leveraged already-collected data and problem structure to perform the optimizations and accelerate training. Our evaluations on real clusters showed how our formulations resulted in improved distributed systems' efficiency and performance, and allowed them to cope with heterogeneity in workloads and resource needs, as well as adapt to time-varying patterns.

Bibliography

[1] Prashanth L. A. and Shalabh Bhatnagar. “Reinforcement Learning With Function Approximation for Traffic Signal Control”. In: *IEEE Trans. Intelligent Transportation Systems* 12.2 (2011), pp. 412–421.

[2] Prashanth L. A. and Shalabh Bhatnagar. “Threshold Tuning Using Stochastic Optimization for Graded Signal Control”. In: *IEEE Trans. Vehicular Technology* 61.9 (2012), pp. 3865–3880.

[3] Prashanth L. A., Abhranil Chatterjee, and Shalabh Bhatnagar. “Adaptive Sleep-Wake Control using Reinforcement Learning in Sensor Networks”. In: *Sixth International Conference on Communication Systems and Networks, COMSNETS 2014, Bangalore, India, January 6-10, 2014*. 2014, pp. 1–8.

[4] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. “TensorFlow: A system for large-scale machine learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283.

[5] Alekh Agarwal et al. “A Reliable Effective Terascale Linear Learning System”. In: *JMLR* 15 (2014).

[6] Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert E. Schapire. "Taming the monster: A fast and simple algorithm for contextual bandits". In: *In Proceedings of the 31st International Conference on Machine Learning (ICML-14*. 2014, pp. 1638–1646.

[7] Rajeev Agrawal. "Sample Mean Based Index Policies with O(log n) Regret for the Multi-Armed Bandit Problem". In: *Advances in Applied Probability* 27.4 (1995), pp. 1054–1078.

[8] Shipra Agrawal and Navin Goyal. "Thompson Sampling for Contextual Bandits with Linear Payoffs". In: *Proceedings of the 30th International Conference on Machine Learning*. Vol. 28. Proceedings of Machine Learning Research 3. PMLR, 2013, pp. 127–135.

[9] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. "Cherrypick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics". In: *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*. NSDI'17. USENIX Association, 2017, pp. 469–482.

[10] *Amazon Web Services Auto Scaling*. https://aws.amazon.com/autoscaling/. Accessed: 11-18-2018.

[11] Moore Andrew. *Reinforcement Learning, Tutorial Slides by Andrew Moore*. https://www.autonlab.org/tutorials/rl.html. Accessed: 11-18-2018.

[12] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. "OpenTuner: An Extensible Framework for Program Autotuning". In: *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*. ACM, 2014, pp. 303–316.

[13] Jason Ansel, Maciej Pacula, Yee Lok Wong, Cy Chan, Marek Olszewski, Una-May O'Reilly, and Saman Amarasinghe. "SiblingRivalry: Online Autotuning Through Local Competitions". In: *Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*. CASES '12. ACM, 2012, pp. 91–100.

[14] Hamid Arabnejad, Claus Pahl, Pooyan Jamshidi, and Giovani Estrada. "A Comparison of Reinforcement Learning Techniques for Fuzzy Cloud Auto-Scaling". In: *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. CCGrid '17. Madrid, Spain: IEEE Press, 2017, pp. 64–73.

[15] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "A Brief Survey of Deep Reinforcement Learning". In: *CoRR* (2017).

[16] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Mach. Learn.* 47.2-3 (2002), pp. 235–256.

[17] Aditya Auradkar et al. "Data infrastructure at linkedIn". In: *ICDE*. 2012.

[18] Jens Axboe. *Flexible I/O Tester*. https://github.com/axboe/fio. 2011.

[19] Haldun Aytug, Siddhartha Bhattacharyya, Gary J. Kochlet, and Jane L. Snowdon. "A Review of Machine Learning in Scheduling". In: *IEEE Transactions on Engineering Management* (1994).

[20] Ricardo Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vassilis Plachouras, and Luca Telloli. "On the Feasibility of Multi-site Web Search Engines". In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. CIKM '09. Hong Kong, China: ACM, 2009, pp. 425–434.

[21] Maria-Florina Balcan, Avrim Blum, Shai Fine, and Yishay Mansour. "Distributed Learning, Communication Complexity and Privacy". In: *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*. 2012, pp. 26.1–26.22.

[22] Hitesh Ballani et al. "Towards Predictable Datacenter Networks". In: *SIGCOMM*. 2011.

[23] Ishan Banerjee, Fei Guo, Kiran Tati, and Rajesh Venkatasubramanian. *Memory Overcommitment in the ESX Server*. 2011.

[24] Sean Kenneth Barker and Prashant Shenoy. "Empirical Evaluation of Latency-sensitive Application Performance in the Cloud". In: *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*. MMSys '10. ACM, 2010, pp. 35–46.

[25] A. Bar-Or, D. Keren, A. Schuster, and R. Wolff. “Hierarchical Decision Tree Induction in Distributed Genomic Databases”. In: *IEEE Transactions on Knowledge and Data Engineering – Special Issue on Mining Biological Data* 17.8 (Aug. 2005).

[26] Daniel S. Berger. “Towards Lightweight and Robust Machine Learning for CDN Caching”. In: *Proceedings of the 17th ACM Workshop on Hot Topics in Networks, HotNets 2018, Redmond, WA, USA, November 15-16, 2018*. 2018, pp. 134–140.

[27] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[28] Alina Beygelzimer, John Langford, Lihong Li, Lev Reyzin, and Robert Schapire. “Contextual Bandit Algorithms with Supervised Learning Guarantees”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. PMLR, Apr. 2011, pp. 19–26.

[29] Norman Bobroff, Andrzej Kochut, and Kirk A. Beaty. “Dynamic Placement of Virtual Machines for Managing SLA Violations.” In: *Integrated Network Management*. IEEE, 2007, pp. 119–128.

[30] Peter Bodik, Rean Griffith, Charles Sutton, Armando Fox, Michael I. Jordan, and David A. Patterson. “Automatic Exploration of Datacenter Performance Regimes”. In: *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*. ACDC ’09. Barcelona, Spain: ACM, 2009, pp. 1–6.

[31] Peter Bodık, Ishai Menache, Mosharaf Chowdhury, Pradeepkumar Mani, David A. Maltz, and Ion Stoica. “Surviving Failures in Bandwidth-constrained Datacenters”. In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM ’12. ACM, 2012, pp. 431–442.

[32] Edward Bortnikov, Ari Frank, Eshcar Hillel, and Sriram Rao. "Predicting Execution Bottlenecks in Map-reduce Clusters". In: *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*. HotCloud'12. USENIX Association, 2012.

[33] Léon Bottou. "Large-scale Machine Learning with Stochastic Gradient Descent". In: *COMPSTAT*. 2010.

[34] Justin A. Boyan and Michael L. Littman. "Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach". In: *Proceedings of the 6th International Conference on Neural Information Processing Systems*. NIPS'93. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1993, pp. 671–678.

[35] Stephen Boyd et al. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Found. Trends Mach. Learn.* 3.1 (Jan. 2011).

[36] Leo Breiman. "Random Forests". In: *Mach. Learn.* 45.1 (2001), pp. 5–32.

[37] X. Bu, J. Rao, and C. Xu. "Coordinated Self-Configuration of Virtual Machines and Appliances Using a Model-Free Learning Approach". In: *IEEE Transactions on Parallel and Distributed Systems* 24.4 (2013), pp. 681–690.

[38] Sébastien Bubeck and Nicolò Cesa-Bianchi. "Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems". In: *Foundations and Trends in Machine Learning* 5.1 (2012), pp. 1–122.

[39] Jacques Bughin, Eric Hazan, Sree Ramaswamy, Michael Chui, Tera Allas, Peter Dahlstrom, Nicolaus Henke, and Monica Trench. *Artificial Intelligence, The Next Digital Frontier*. 2017.

[40] Christopher J. C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". In: *Data Min. Knowl. Discov.* 2.2 (1998), pp. 121–167.

[41] Berkant Barla Cambazoglu and Ricardo Baeza-Yates. "Scalability Challenges in Web Search Engines". In: (2011). Ed. by Massimo Melucci and Ricardo Baeza-Yates, pp. 27–50.

[42] Ignacio Cano, Srinivas Aiyar, Varun Arora, Manosiz Bhattacharyya, Akhilesh Chaganti, Chern Cheah, Brent N. Chun, Karan Gupta, Vinayak Khot, and Arvind Krishnamurthy. "Curator: Self-Managing Storage for Enterprise Clusters". In: *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*. 2017, pp. 51–66.

[43] Ignacio Cano, Srinivas Aiyar, and Arvind Krishnamurhty. "Characterizing Private Clouds: A Large-Scale Empirical Analysis of Enterprise Clusters". In: *Proceedings of the Seventh ACM Symposium on Cloud Computing*. SoCC '16. Santa Clara, CA, USA: ACM, 2016, pp. 29–41.

[44] Ignacio Cano, Lequn Chen, Pedro Fonseca, Tianqi Chen, Chern Cheah, Karan Gupta, Ramesh Chandra, and Arvind Krishnamurthy. "ADARES: Adaptive Resource Management for Virtual Machines". In: *CoRR* (2018).

[45] Ignacio Cano, Markus Weimer, Dhruv Mahajan, Carlo Curino, and Giovanni Matteo Fumarola. "Towards Geo-Distributed Machine Learning". In: *CoRR* abs/1603.09035 (2016). URL: http://arxiv.org/abs/1603.09035.

[46] Ignacio Cano, Markus Weimer, Dhruv Mahajan, Carlo Curino, and Giovanni Matteo Fumarola. "Towards Geo-Distributed Machine Learning". In: *Learning Systems Workshop at NIPS 2015*. 2015.

[47] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. "AuTO: Scaling Deep Reinforcement Learning for Datacenter-scale Automatic Traffic Optimization". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '18. ACM, 2018, pp. 191–205.

[48] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems". In: *CoRR* (2015).

[49] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. "Project Adam: Building an Efficient and Scalable Deep Learning Training System". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, 2014, pp. 571–582.

[50] Cheng-tao Chu et al. "Map-Reduce for Machine Learning on Multicore". In: *NIPS*. 2007.

[51] Apache Hadoop community. "Scaling out YARN via Federation". In: https://issues.apache.org/jira/browse/YARN-2915. 2016.

[52] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Mach. Learn.* 20.3 (1995), pp. 273–297.

[53] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. "Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms". In: *Proceedings of the 26th ACM Symposium on Operating Systems Principles*. 2017.

[54] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. "Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. ACM, 2017, pp. 153–167.

[55] *CPU hotplug in the Kernel*. https://www.kernel.org/doc/html/v4.14/core-api/cpu_hotplug.html. Accessed: 11-19-2018.

[56] Criteo Labs. "Terabyte Click Logs". In: (2015). http://labs.criteo.com/downloads/download-terabyte-click-logs/. Accessed 11-19-2018.

[57] C. Cummins, P. Petoumenos, Z. Wang, and H. Leather. "End-to-End Deep Learning of Optimization Heuristics". In: *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 2017, pp. 219–232.

[58] Peter Dayan. "The Convergence of TD(λ) for General λ". In: *Machine Learning* 8 (1992), pp. 341–362.

[59] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. In: *Commun. ACM* 51.1 (2008), pp. 107–113.

[60] *DeepMind AI Reduces Google Data Centre Cooling Bill by 40%*. https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/. Accessed 11-12-2018. 2016.

[61] Ofer Dekel. “From Online to Batch Learning with Cutoff-Averaging”. In: *Advances in Neural Information Processing Systems 21*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou. Curran Associates, Inc., 2009, pp. 377–384.

[62] Ofer Dekel and Yoram Singer. “Data-Driven Online to Batch Conversions”. In: *Advances in Neural Information Processing Systems 18*. Ed. by Y. Weiss, B. Schölkopf, and J. C. Platt. MIT Press, 2006, pp. 267–274.

[63] Christina Delimitrou. “Improving Resource Efficiency in Cloud Computing”. PhD thesis. Stanford University, 2015.

[64] Christina Delimitrou and Christos Kozyrakis. “HCloud: Resource-Efficient Provisioning in Shared Cloud Systems”. In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’16. Atlanta, Georgia, USA: ACM, 2016, pp. 473–488.

[65] Christina Delimitrou and Christos Kozyrakis. “Paragon: QoS-aware Scheduling for Heterogeneous Datacenters”. In: *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’13. Houston, Texas, USA: ACM, 2013, pp. 77–88.

[66] Christina Delimitrou and Christos Kozyrakis. “Quasar: Resource-efficient and QoS-aware Cluster Management”. In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’14. Salt Lake City, Utah, USA: ACM, 2014, pp. 127–144.

[67] *Demystifying GDPR: Separating fact from fiction*. https://venturebeat.com/2018/05/20/demystifying-gdpr-separating-fact-from-fiction/. Accessed: 11-21-2018.

[68] John Dunagan, Alice X. Zheng, and Daniel R. Simon. "Heat-ray: Combating Identity Snowball Attacks Using Machinelearning, Combinatorial Optimization and Attack Graphs". In: *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*. SOSP '09. ACM, 2009, pp. 305–320.

[69] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. "Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: towards a fully automated workflow". In: *7th International Conference on Autonomic and Autonomous Systems*. May 2011, pp. 67–74.

[70] Xavier Dutreilh, Aurélien Moreau, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. "From Data Center Resource Allocation to Control Theory and Back". In: *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*. CLOUD '10. IEEE Computer Society, 2010, pp. 410–417.

[71] Jonathan Eastep, David Wingate, and Anant Agarwal. "Smart Data Structures: An Online Machine Learning Approach to Multicore Data Structures". In: *Proceedings of the 8th International Conference on Autonomic Computing, ICAC 2011, Karlsruhe, Germany, June 14-18, 2011*. 2011, pp. 11–20.

[72] Jonathan Eastep, David Wingate, Marco D. Santambrogio, and Anant Agarwal. "Smartlocks: Lock Acquisition Scheduling for Self-Aware Synchronization". In: *Proceedings of the 7th International Conference on Autonomic Computing*. 2010.

[73] John K. Edwards, Daniel Ellard, Craig Everhart, Robert Fair, Eric Hamilton, Andy Kahn, Arkady Kanevsky, James Lentini, Ashish Prakash, Keith A. Smith, and Edward Zayas. "FlexVol: Flexible, Efficient File Volume Virtualization in WAFL". In: *USENIX 2008 Annual Technical Conference*. ATC'08. Boston, Massachusetts: USENIX Association, 2008, pp. 129–142.

[74] EMC. *EMC Isilon OneFS: A Technical Overview*. 2016.

[75] European Commission press release. *Commission to pursue role as honest broker in future global negotiations on Internet Governance*. http://europa.eu/rapid/press-release_IP-14-142_en.htm. Accessed 11-19-2018.

[76] Benjamin Farley, Ari Juels, Venkatanathan Varadarajan, Thomas Ristenpart, Kevin D. Bowers, and Michael M. Swift. "More for Your Money: Exploiting Performance Heterogeneity in Public Clouds". In: *Proceedings of the Third ACM Symposium on Cloud Computing*. SoCC '12. San Jose, California: ACM, 2012.

[77] Alexandra Fedorova, David Vengerov, and Daniel Doucette. "Operating system Scheduling on Heterogeneous Core Systems". In: *Proceedings of 2007 Operating System Support for Heterogeneous Multicore Architectures*. 2007.

[78] V. Feldman. "A Complete Characterization of Statistical Query Learning with Applications to Evolvability". In: *2009 50th Annual IEEE Symposium on Foundations of Computer Science(FOCS)*. Oct. 2010, pp. 375–384.

[79] Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. "Jockey: Guaranteed Job Latency in Data Parallel Clusters". In: *Proceedings of the 7th ACM European Conference on Computer Systems*. EuroSys '12. ACM, 2012, pp. 99–112.

[80] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Y. Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. "Reducing Web Latency: The Virtue of Gentle Aggression". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. ACM, 2013, pp. 159–170.

[81] Guillem Francès, Xiao Bai, B. Barla Cambazoglu, and Ricardo Baeza-Yates. "Improving the Efficiency of Multi-site Web Search Engines". In: *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. WSDM '14. New York, New York, USA: ACM, 2014, pp. 3–12.

[82] Jerome H. Friedman. “Greedy Function Approximation: A Gradient Boosting Machine”. In: *Annals of Statistics* 29 (2000), pp. 1189–1232.

[83] Errin W. Fulp, Glenn A. Fink, and Jereme N. Haack. “Predicting Computer System Failures Using Support Vector Machines”. In: *Proceedings of the First USENIX Conference on Analysis of System Logs*. WASL’08. USENIX Association, 2008.

[84] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L. Wiener, Armando Fox, Michael Jordan, and David Patterson. “Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning”. In: *Proceedings of the 2009 IEEE International Conference on Data Engineering*. ICDE ’09. IEEE Computer Society, 2009, pp. 592–603.

[85] Javier Garcıa and Fernando Fernández. “A Comprehensive Survey on Safe Reinforcement Learning”. In: *J. Mach. Learn. Res.* (2015), pp. 1437–1480.

[86] Gartner. *Gartner Says By 2020, Artificial Intelligence Will Create More Jobs Than It Eliminates*. https://www.gartner.com/newsroom/id/3837763. Accessed 11-9-2018. 2017.

[87] *GDPR goes live this week. What happens next?* https://venturebeat.com/2018/05/22/gdpr-goes-live-tomorrow-what-happens-next/. Accessed: 11-21-2018.

[88] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google File System”. In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. SOSP ’03. Bolton Landing, NY, USA: ACM, 2003, pp. 29–43.

[89] Gluster. *Cloud Storage for the Modern Data Center: An Introduction to Gluster Architecture*. 2011.

[90] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. “Workload Analysis and Demand Prediction of Enterprise Data Center Applications”. In: *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*. IISWC ’07. IEEE Computer Society, 2007, pp. 171–180.

[91] Anna Goldie, Azalia Mirhoseini, Jonathan Raiman, Kevin Swersky, and Milad Hashemi. *ML4Systems Workshop*. http://mlforsystems.org/. Accessed 11-9-2018. 2018.

[92] Zhenhuan Gong, Xiaohui Gu, and J. Wilkes. “PRESS: PRedictive Elastic ReSource Scaling for cloud systems”. In: *2010 International Conference on Network and Service Management*. 2010, pp. 9–16.

[93] *Google Cloud Platform AutoScaler*. https://cloud.google.com/compute/docs/autoscaler/. Accessed: 11-10-2018.

[94] Sriram Govindan, Jeonghwan Choi, Bhuvan Urgaonkar, Anand Sivasubramaniam, and Andrea Baldini. “Statistical Profiling-based Techniques for Effective Power Provisioning in Data Centers”. In: *Proceedings of the 4th ACM European Conference on Computer Systems*. EuroSys ’09. ACM, 2009, pp. 317–330.

[95] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. “The Cost of a Cloud: Research Problems in Data Center Networks”. In: *SIGCOMM Comput. Commun. Rev.* 39.1 (Dec. 2008), pp. 68–73.

[96] William D. Gropp et al. *MPI : the complete reference. Vol. 2. , The MPI-2 extensions*. 1998.

[97] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. “Learning Memory Access Patterns”. In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1919–1928.

[98] Fabien Hermenier, Julia Lawall, and Gilles Muller. “BtrPlace: A Flexible Consolidation Manager for Highly Available Applications”. In: *IEEE Trans. Dependable Secur. Comput.* 10.5 (2013), pp. 273–286.

[99] Benjamin Hindman et al. “Mesos: A Platform for Fine-grained Resource Sharing in the Data Center”. In: *NSDI*. 2011.

[100] Henry Hoffmann. “JouleGuard: energy guarantees for approximate applications.” In: *SOSP*. Ed. by Ethan L. Miller and Steven Hand. ACM, 2015, pp. 198–214.

[101] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. “Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds”. In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, 2017, pp. 629–647.

[102] Alexandru Iosup, Nezih Yigitbasi, and Dick Epema. “On the Performance Variability of Production Cloud Services”. In: *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. CCGRID ’11. IEEE Computer Society, 2011, pp. 104–113.

[103] Engin Ipek, Onur Mutlu, José Martınez, and Rich Caruana. “Self-Optimizing Memory Controllers: A Reinforcement Learning Approach”. In: *Proceedings of the 35th Annual International Symposium on Computer Architecture*. ISCA ’08. IEEE Computer Society, 2008, pp. 39–50.

[104] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. “Quincy: Fair Scheduling for Distributed Computing Clusters”. In: *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*. SOSP ’09. Big Sky, Montana, USA: ACM, 2009, pp. 261–276.

[105] Martin Jaggi et al. “Communication-Efficient Distributed Dual Coordinate Ascent”. In: *NIPS*. 2014.

[106] Kevin G Jamieson, Lalit Jain, Chris Fernandez, Nicholas J. Glattard, and Rob Nowak. “NEXT: A System for Real-World Development, Evaluation, and Application of Active Learning”. In: *Advances in Neural Information Processing Systems 28*. 2015, pp. 2656–2664.

[107] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. “Pytheas: Enabling Data-driven Quality of Experience Optimization Using Group-based Exploration-exploitation”. In: *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*. NSDI’17. Boston, MA, USA: USENIX Association, 2017, pp. 393–406.

[108] Court of Justice of the European Union. “The Court of Justice declares that the Commission’s US Safe Harbour Decision is invalid”. In: (2015). http://g8fip1kplyr33r3krz5b97d1.wpengine.netdna-cdn.com/wp-content/uploads/2015/10/schrems-judgment.pdf. Accessed 19-11-2018.

[109] Tomer Kaftan, Magdalena Balazinska, Alvin Cheung, and Johannes Gehrke. “Cuttlefish: A Lightweight Primitive for Adaptive Query Processing”. In: *CoRR* abs/1802.09180 (2018).

[110] Kaggle Criteo Labs. “Display Advertising Challenge”. In: (2014). https://www.kaggle.com/c/criteo-display-ad-challenge. Accessed 11-19-2018.

[111] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. “Self-adaptive and Self-configured CPU Resource Provisioning for Virtualized Servers Using Kalman Filters”. In: *Proceedings of the 6th International Conference on Autonomic Computing*. ICAC ’09. ACM, 2009, pp. 117–126.

[112] Ken Kansky, Tom Silver, David A. Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, D. Scott Phoenix, and Dileep George. “Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017, pp. 1809–1818.

[113] Enver Kayaaslan, B. Barla Cambazoglu, and Cevdet Aykanat. “Document replication strategies for geographically distributed web search engines”. In: 49.1 (2013), pp. 51–66. ISSN: 0306-4573.

[114] Michael Kearns. “Efficient Noise-tolerant Learning from Statistical Queries”. In: *J. ACM* 45.6 (Nov. 1998).

[115] Cinar Kilcioglu, Justin M. Rao, Aadharsh Kannan, and R. Preston McAfee. “Usage Patterns and the Economics of the Public Cloud”. In: *Proceedings of the Twenty-Sixth International World Wide Web Conference*. 2017.

[116] Jakub Konecný, Brendan McMahan, and Daniel Ramage. "Federated Optimization: Distributed Optimization Beyond the Datacenter". In: *CoRR* (2015). URL: http://arxiv.org/abs/1511.03575.

[117] Jakub Konecný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. "Federated Optimization: Distributed Machine Learning for On-Device Intelligence". In: *CoRR* (2016). URL: http://arxiv.org/abs/1610.02527.

[118] Jakub Konecný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. "Federated Learning: Strategies for Improving Communication Efficiency". In: *CoRR* abs/1610.05492 (2016). URL: http://arxiv.org/abs/1610.05492.

[119] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. "The Case for Learned Index Structures". In: *Proceedings of the 2018 International Conference on Management of Data*. SIGMOD '18. ACM, 2018, pp. 489–504.

[120] Andreas Krause and Daniel Golovin. "Submodular Function Maximization." In: *Tractability*. Ed. by Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli. Cambridge University Press, 2014, pp. 71–104.

[121] Andreas Krause, Ram Rajagopal, Anupam Gupta, and Carlos Guestrin. "Simultaneous Placement and Scheduling of Sensors". In: *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*. IPSN '09. IEEE Computer Society, 2009, pp. 181–192.

[122] Poole David L. and Mackworth Alan K. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, 2010.

[123] Avinash Lakshman and Prashant Malik. "Cassandra: A Decentralized Structured Storage System". In: *SIGOPS Oper. Syst. Rev.* 44 (2010), pp. 35–40.

[124] Leslie Lamport. "Paxos Made Simple". In: *ACM SIGACT News*. Vol. 32. 4. 2001, pp. 51–58.

[125] John Langford and Tong Zhang. "The Epoch-Greedy Algorithm for Multi-armed Bandits with Side Information". In: *Advances in Neural Information Processing Systems 20*. Ed. by J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis. Curran Associates, Inc., 2008, pp. 817–824. URL: http://papers.nips.cc/paper/3178-the-epoch-greedy-algorithm-for-multi-armed-bandits-with-side-information.pdf.

[126] Nikolaos Laoutaris et al. "Inter-datacenter bulk transfers with netstitcher". In: *SIGCOMM*. 2011.

[127] Kim Larry. *How Many Ads Does Google Serve In A Day?* http://goo.gl/oIidXO. Accessed 11-13-2018. 2012.

[128] Arnon Lazerson, Izchak Sharfman, Daniel Keren, Assaf Schuster, Minos Garofalakis, and Vasilis Samoladas. "Monitoring Distributed Streams Using Convex Decompositions". In: *Proc. VLDB Endow.* 8.5 (Jan. 2015), pp. 545–556. ISSN: 2150-8097. DOI: 10.14778/2735479.2735487. URL: http://dx.doi.org/10.14778/2735479.2735487.

[129] Mathias Lecuyer, Joshua Lockerman, Lamont Nelson, Siddhartha Sen, Amit Sharma, and Aleksandrs Slivkins. "Harvesting Randomness to Optimize Distributed Systems". In: *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. HotNets-XVI. ACM, 2017, pp. 178–184.

[130] George Lee et al. "The Unified Logging Infrastructure for Data Analytics at Twitter". In: *PVLDB* (2012).

[131] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. "A Contextual-bandit Approach to Personalized News Article Recommendation". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: ACM, 2010, pp. 661–670.

[132] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. "Scaling Distributed Machine

Learning with the Parameter Server”. In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, 2014, pp. 583–598.

[133] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. “Communication Efficient Distributed Machine Learning with the Parameter Server”. In: *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014, pp. 19–27.

[134] Yuanlong Li, Yonggang Wen, Kyle Guan, and Dacheng Tao. “Transforming Cooling Optimization for Green Data Center via Deep Reinforcement Learning”. In: *CoRR* (2017).

[135] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. “Continuous control with deep reinforcement learning”. In: *CoRR* (2015).

[136] Chih-Jen Lin et al. “Trust Region Newton Method for Logistic Regression”. In: *J. Mach. Learn. Res.* 9 (2008).

[137] Yucheng Low et al. “Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud”. In: *PVLDB* (2012).

[138] Chenyang Lu, John Stankovic, Sang Son, and Gang Tao. “Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms”. In: *Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing* 23 (2002).

[139] Dhruv Mahajan et al. “A Functional Approximation Based Distributed Learning Algorithm”. In: *CoRR* (2013).

[140] Dhruv Mahajan et al. “An efficient distributed learning algorithm based on effective local functional approximations”. In: (). Arxiv http://arxiv.org/abs/1310.8418v4.

[141] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. “Resource Management with Deep Reinforcement Learning”. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. HotNets ’16. ACM, 2016, pp. 50–56.

[142] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. "Neural Adaptive Video Streaming with Pensieve". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '17. ACM, 2017, pp. 197–210.

[143] Benedict C. May, Nathan Korda, Anthony Lee, and David S. Leslie. "Optimistic Bayesian Sampling in Contextual-bandit Problems". In: *J. Mach. Learn. Res.* 13.1 (2012).

[144] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*. 2017, pp. 1273–1282.

[145] *Memory Hotplug*. https://github.com/spotify/linux/blob/master/Documentation/memory-hotplug.txt. Accessed: 11-19-2018.

[146] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. "MLlib: Machine Learning in Apache Spark". In: *J. Mach. Learn. Res.* 17.1 (2016), pp. 1235–1241.

[147] A. Mirhoseini, H. Pham, Q. Le, M. Norouzi, S. Bengio, B. Steiner, Y. Zhou, N. Kumar, R. Larsen, and J. Dean. "Device Placement Optimization with Reinforcement Learning". In: 2017.

[148] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le, and Jeff Dean. "A Hierarchical Model for Device Placement". In: *International Conference on Learning Representations*. 2018.

[149] Asit K. Mishra, Joseph L. Hellerstein, Walfredo Cirne, and Chita R. Das. "Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters". In: *SIGMETRICS Perform. Eval. Rev.* 37.4 (2010), pp. 34–41.

[150] Thomas Mitchell. *Machine Learning*. 1st ed. McGraw-Hill, 1997.

[151] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. “Asynchronous Methods for Deep Reinforcement Learning”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Vol. 48. Proceedings of Machine Learning Research. PMLR, 20–22 Jun 2016, pp. 1928–1937.

[152] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. “Playing Atari With Deep Reinforcement Learning”. In: *NIPS Deep Learning Workshop*. 2013.

[153] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. “Human-level Control through Deep Reinforcement Learning”. In: *Nature* 518.7540 (2015), pp. 529–533.

[154] L. Mönch, J. Zimmermann, and P. Otto. “Machine Learning Techniques for Scheduling Jobs with Incompatible Families and Unequal Ready Times on Parallel Batch Machines”. In: *Eng. Appl. Artif. Intell.* 19.3 (2006), pp. 235–245.

[155] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[156] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. “Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning”. In: *CoRR* abs/1708.02596 (2017).

[157] Shravan Narayanamurthy et al. “Towards Resource-Elastic Machine Learning”. In: *Workshop on Big Learning, NIPS*. 2013.

[158] Dejan Novaković, Nedeljko Vasić, Stanko Novaković, Dejan Kostić, and Ricardo Bianchini. “DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments”. In: *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*. San Jose, CA: USENIX Association, 2013, pp. 219–230.

[159] Dirk Ormoneit and Saunak Sen. “Kernel-Based Reinforcement Learning”. In: *Machine Learning*. 1999, pp. 161–178.

[160] Pradeep Padala, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. “Adaptive Control of Virtualized Resources in Utility Computing Environments”. In: *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. EuroSys ’07. ACM, 2007, pp. 289–302.

[161] Sinno Jialin Pan and Qiang Yang. “A Survey on Transfer Learning”. In: *IEEE Trans. on Knowl. and Data Eng.* 22.10 (Oct. 2010), pp. 1345–1359.

[162] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[163] Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. “A survey of methods for distributed machine learning”. In: *Progress in AI* (2013), pp. 1–11.

[164] Barry Porter, Matthew Grieves, Roberto Rodrigues Filho, and David Leslie. “REX: A Development Platform and Online Learning Approach for Runtime Emergent Software Systems”. In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI’16. Savannah, GA, USA: USENIX Association, 2016, pp. 333–348.

[165] Paolo Priore, David De La Fuente, Alberto Gomez, and Javier Puente. “A Review of Machine Learning in Dynamic Scheduling of Flexible Manufacturing Systems”. In: *Artif. Intell. Eng. Des. Anal. Manuf.* 15.3 (2001), pp. 251–263.

[166] Paolo Priore, David de la Fuente, Javier Puente, and José Parreño. “A Comparison of Machine-learning Algorithms for Dynamic Scheduling of Flexible Manufacturing Systems”. In: *Eng. Appl. Artif. Intell.* 19.3 (2006), pp. 247–255.

[167] Qifan Pu et al. “Low Latency, Geo-distributed Data Analytics”. In: *SIGCOMM*. 2015.

[168] Larry D. Pyeatt and Adele E. Howe. *Decision Tree Function Approximation in Reinforcement Learning*. Tech. rep. Proceedings of the Third International Symposium on Adaptive Systems: Evolutionary Computation and Probabilistic Graphical Models, 1998.

[169] Ariel Rabkin et al. “Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area”. In: *NSDI*. 2014.

[170] Bogdan Răducanu, Peter Boncz, and Marcin Zukowski. “Micro Adaptivity in Vectorwise”. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’13. ACM, 2013, pp. 1231–1242.

[171] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. “VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration”. In: *Proceedings of the 6th International Conference on Autonomic Computing*. ICAC ’09. ACM, 2009, pp. 137–146.

[172] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. “Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis”. In: *Proceedings of the Third ACM Symposium on Cloud Computing*. SoCC ’12. San Jose, California: ACM, 2012, 7:1–7:13.

[173] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. *Google cluster-usage traces: format + schema*. Technical Report. Google Inc., Nov. 2011.

[174] RightScale. *State of the Cloud Report*. https://www.rightscale.com/lp/state-of-the-cloud. 2018.

[175] Ohad Rodeh and Avi Teperman. “zFS - A Scalable Distributed File System Using Object Disks”. In: *IEEE Symposium on Mass Storage Systems*. IEEE Computer Society, 2003, pp. 207–218.

[176] Martin Rost and Kirsten Bock. “Privacy by design and the new protection goals”. In: *DuD, January* (2011).

[177] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. "VM Live Migration At Scale". In: *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. VEE '18. ACM, 2018.

[178] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[179] S. Ratna Sandeep, M. Swapna, Thirumale Niranjan, Sai Susarla, and Siddhartha Nandi. "CLUEBOX: A Performance Log Analyzer for Automated Troubleshooting". In: *Proceedings of the First USENIX Conference on Analysis of System Logs*. WASL'08. USENIX Association, 2008.

[180] Robert R. Schaller. "Moore's Law: Past, Present, and Future". In: *IEEE Spectr.* 34.6 (1997), pp. 52–59.

[181] Frank Schmuck and Roger Haskin. "GPFS: A Shared-Disk File System for Large Computing Clusters". In: *Proceedings of the 1st USENIX Conference on File and Storage Technologies*. FAST '02. Monterey, CA: USENIX Association, 2002.

[182] Malte Schwarzkopf et al. "Omega: flexible, scalable schedulers for large compute clusters". In: *EuroSys*. 2013.

[183] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report. University of Wisconsin–Madison, 2009.

[184] Sreekanth Setty. *VMware vSphere 5.1 vMotion Architecture, Performance and Best Practices*. 2012.

[185] Jonathan R Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep. 1994.

[186] Piyush Shivam, Shivnath Babu, and Jeff Chase. "Active and Accelerated Learning of Cost Models for Optimizing Scientific Applications". In: *Proceedings of the 32Nd International Conference on Very Large Data Bases*. VLDB '06. VLDB Endowment, 2006, pp. 535–546.

[187] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. “The Hadoop Distributed File System”. In: *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. MSST ’10. IEEE Computer Society, 2010, pp. 1–10.

[188] Tom Simonite. *Moore’s Law Is Dead. Now What?* https://www.technologyreview.com/s/601441/moores-law-is-dead-nowwhat/. Accessed 11-9-2018. 2016.

[189] Virginia Smith. “System-Aware Optimization for Machine Learning at Scale”. PhD thesis. EECS Department, University of California, Berkeley, 2017. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-140.html.

[190] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. “Federated Multi-Task Learning”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. 2017, pp. 4424–4434.

[191] Virginia Smith, Simone Forte, Chenxin Ma, Martin Takáč, Michael I. Jordan, and Martin Jaggi. “CoCoA: A General Framework for Communication-Efficient Distributed Optimization”. In: *Journal of Machine Learning Research* 18.230 (2018), pp. 1–49.

[192] Ahmed A. Soror, Umar Farooq Minhas, Ashraf Aboulnaga, Kenneth Salem, P. Kokosielis, and Sunil Kamath. “Automatic Virtual Machine Configuration for Database Workloads”. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’08. ACM, 2008, pp. 953–966.

[193] C. Spearman. “The Proof and Measurement of Association between Two Things”. In: *The American Journal of Psychology* 15.1 (1904), pp. 72–101. ISSN: 00029556. URL: http://www.jstor.org/stable/1412159.

[194] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias W. Seeger. “Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design”. In: *ICML*. 2010, pp. 1015–1022.

[195] Christopher Stewart, Terence Kelly, Alex Zhang, and Kai Shen. “A Dollar from 15 Cents: Cross-platform Management for Internet Services”. In: *USENIX 2008 Annual Technical Conference*. ATC’08. USENIX Association, 2008, pp. 199–212.

[196] Sun. *Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System*. 2007.

[197] Ananda Theertha Suresh, Felix X. Yu, H. Brendan McMahan, and Sanjiv Kumar. “Distributed Mean Estimation with Limited Communication”. In: *CoRR* abs/1611.00429 (2016).

[198] Richard S. Sutton. “Learning to Predict by the Methods of Temporal Differences”. In: *MACHINE LEARNING*. Kluwer Academic Publishers, 1988, pp. 9–44.

[199] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[200] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS’99. Denver, CO: MIT Press, 1999, pp. 1057–1063.

[201] Richard Stuart Sutton. “Temporal Credit Assignment in Reinforcement Learning”. PhD thesis. 1984.

[202] B. C. Tak, C. Tang, H. Huang, and L. Wang. “PseudoApp: Performance prediction for application migration to cloud”. In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. 2013, pp. 303–310.

[203] Ambuj Tewari and Susan A. Murphy. *From Ads to Interventions: Contextual Bandits in Mobile Health*. 2017.

[204] *The Nutanix Bible*. http://nutanixbible.com/. Accessed: 11-28-2018.

[205] *The Upper Confidence Bound Algorithm*. http://banditalgs.com/2016/09/18/the-upper-confidence-bound-algorithm/. Accessed: 09-15-2018.

[206] Ashish Thusoo et al. “Data Warehousing and Analytics Infrastructure at Facebook”. In: SIGMOD. 2010.

[207] John N. Tsitsiklis and Benjamin Van Roy. *An Analysis of Temporal-Difference Learning with Function Approximation*. Tech. rep. IEEE Transactions on Automatic Control, 1997.

[208] Bhuvan Urgaonkar, Prashant Shenoy, and Timothy Roscoe. “Resource Overbooking and Application Profiling in Shared Hosting Platforms”. In: *SIGOPS Oper. Syst. Rev.* 36 (2002), pp. 239–254.

[209] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. “Automatic Database Management System Tuning Through Large-scale Machine Learning”. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. SIGMOD ’17. ACM, 2017, pp. 1009–1024.

[210] Nedeljko Vasić, Dejan Novaković, Svetozar Miučin, Dejan Kostić, and Ricardo Bianchini. “DejaVu: Accelerating Resource Allocation in Virtualized Environments”. In: *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XVII. London, England, UK: ACM, 2012, pp. 423–436.

[211] Vinod Kumar Vavilapalli et al. “Apache Hadoop YARN: Yet Another Resource Negotiator”. In: *SOCC*. 2013.

[212] *Vdbench*. https://www.oracle.com/technetwork/server-storage/vdbench-downloads-1901681.html. Accessed: 11-19-2018.

[213] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. “Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics”. In: *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, 2016, pp. 363–378.

[214] VMware. *Performance Best Practices for VMware vSphere 5.5*. 2014.

[215] VMware. *Performance Best Practices for VMware vSphere 6.0*. 2015.

[216] VMware. *Understanding Memory Resource Management in VMware ESX Server*. 2011.

[217] Ashish Vulimiri et al. "Global Analytics in the Face of Bandwidth and Regulatory Constraints". In: *NSDI*. 2015.

[218] Ashish Vulimiri. "Latency-Bandwidth Tradeoff in Internet Applications". PhD thesis. 2015.

[219] Ashish Vulimiri et al. "WANalytics: Analytics for a Geo-Distributed Data-Intensive World". In: *CIDR* (2015).

[220] Christopher J. C. H. Watkins and Peter Dayan. "Technical Note: Q-Learning". In: *Mach. Learn.* 8.3-4 (1992), pp. 279–292.

[221] Markus Weimer et al. "REEF: Retainable Evaluator Execution Framework". In: *SIGMOD*. 2015.

[222] K.Q. Weinberger et al. "Feature hashing for large scale multitask learning". In: *ICML*. 2009.

[223] Shimon Whiteson and Peter Stone. "Adaptive Job Routing and Scheduling". In: *Eng. Appl. Artif. Intell.* 17.7 (Oct. 2004), pp. 855–869.

[224] John Wilkes. *More Google Cluster Data*. http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html. Accessed 11-19-2018. 2011.

[225] Eric P. Xing, Qirong Ho, Wei Dai, Jin-Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. "Petuum: A New Platform for Distributed Machine Learning on Big Data". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '15. ACM, 2015, pp. 1335–1344.

[226] Neeraja Yadwadkar. *Machine Learning for Automatic Resource Management in the Datacenter and the Cloud*. Tech. rep. EECS Department, University of California, Berkeley, 2018.

[227] Neeraja J. Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. “Wrangler: Predictable and Faster Jobs Using Fewer Resources”. In: *Proceedings of the ACM Symposium on Cloud Computing*. SOCC ’14. Seattle, WA, USA: ACM, 2014, 26:1–26:14.

[228] Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, Burton Smith, and Randy H. Katz. “Selecting the Best VM Across Multiple Public Clouds: A Data-driven Performance Modeling Approach”. In: *Proceedings of the 2017 Symposium on Cloud Computing*. SoCC ’17. ACM, 2017.

[229] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. “Bubble-flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers”. In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ISCA ’13. Tel-Aviv, Israel: ACM, 2013, pp. 607–618.

[230] Yuehwern Yih. “Learning Real-Time Scheduling Rules from Optimal Policy of Semi-Markov Decision Processes”. In: *International Journal of Computer Integrated Manufacturing* (1992).

[231] Matei Zaharia et al. “Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing”. In: *NSDI*. 2012.

[232] Caoxie Zhang et al. “Efficient Distributed Linear Classification Algorithms via the Alternating Direction Method of Multipliers”. In: *AISTATS*. 2012.

[233] Yuchen Zhang, Martin J Wainwright, and John C Duchi. “Communication-Efficient Algorithms for Statistical Optimization”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1502–1510.

[234] Wei Zheng, Ricardo Bianchini, G. John Janakiraman, Jose Renato Santos, and Yoshio Turner. “JustRunIt: Experiment-based Management of Virtualized Data Centers”. In: *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*. USENIX’09. USENIX Association, 2009, pp. 18–18.

[235] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova. “1000 Islands: Integrated Capacity and Workload Management for the Next Generation Data Center”. In: *2008 International Conference on Autonomic Computing*. 2008, pp. 172–181.

[236] Xiaoyun Zhu, Donald Young, Brian J. Watson, Zhikui Wang, Jerry Rolia, Sharad Singhal, Bret Mckee, Chris Hyser, Daniel Gmach, Robert Gardner, Tom Christian, and Ludmila Cherkasova. “1000 Islands: An Integrated Approach to Resource Management for Virtualized Data Centers”. In: *Cluster Computing* 12.1 (2009), pp. 45–57.

[237] Marcin Zukowski and Peter A. Boncz. “Vectorwise: Beyond Column Stores”. In: *IEEE Data Eng. Bull.* 35.1 (2012), pp. 21–27.