AdaRes: Adaptive Resource Management for Virtual Machines

Ignacio Cano[w] Tianqi Chen[w] Pedro Foncesa[p] Chern Cheah[n]
Karan Gupta[n] Ramesh Chandra[n] Arvind Krishnamurthy[w]

[w]Paul G. Allen School of Computer Science & Engineering, University of Washington
{icano,tqchen,arvind}@cs.washington.edu
[p]Department of Computer Science, Purdue University
pfonseca@purdue.edu
[n]Nutanix Inc.
{chern,karan.gupta,ramesh.chandra}@nutanix.com

Abstract

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 paper, 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 dynamically adjusts VM resources using machine learning techniques. In particular, AdaRes leverages the *contextual bandits* framework to effectively manage the adaptations. Our system exploits easily collectible data, at the cluster, node, and VM levels, to make more sensible allocation decisions, and uses *transfer learning* to safely explore the configurations space and speed up training. Our 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.

1 Introduction

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 [51].

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 [12, 24, 21, 28], 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 [55, 53, 11, 54]. This motivates the need for a system that adaptively changes the amount of system resources allocated to each VM in a cluster.

In this paper, 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 a commercial computation and storage virtualization product. 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 [17, 40, 56, 34], 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 [42]. 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.[1]

Based on our findings, we design and build AdaRes, an adaptive system that automatically optimizes VM re-

[1]We will make publicly available the complete traces that we collected for this study to enable others to build on our work.

source allocations in real clusters. ADARES uses the multi-armed bandit framework with context information [32], also known as contextual bandits, to dynamically tune the VMs resource settings, 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 [7, 33, 13, 32].

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 pre-train (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 [38, 25], which translates into up to 2x 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.

Summarizing, our main contributions 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 build ADARES, an adaptive system capable of tuning VM resources to increase overall system efficiency that is compatible with existing cluster schedulers.
- We propose a contextual bandit-based approach to drive the resource adjustments, and we instantiate our model with an appropriate formulation that results in better resource allocations in real clusters, with resource savings up to 35% (CPU) and 60% (memory) in synthetic workloads executed on real clusters, when compared to threshold and other ML-based baselines.

2 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.

2.1 Measurement Methodology

We perform our measurements on enterprise clusters running a commercial virtual execution platform.[2] Its 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). Our 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 1 shows an overview of the virtual execution environments that we study, containing more than 250K VM traces.

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

Table 1: Dataset Overview

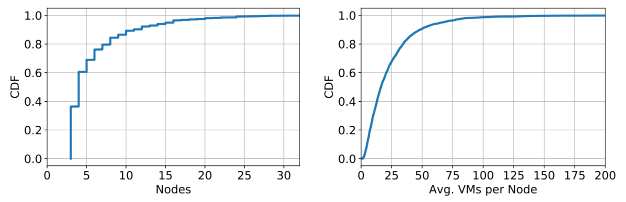
2.2 Private Cluster Configurations

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 1 shows the distribution of nodes per cluster (1a) and the consolidation factor, i.e., the average number of VMs per node, (1b). From Figure 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 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 $\sim$200 VMs per node.

Node-level Configuration Enterprise clusters often have powerful nodes, as shown in Figure 2. We observe that 50%

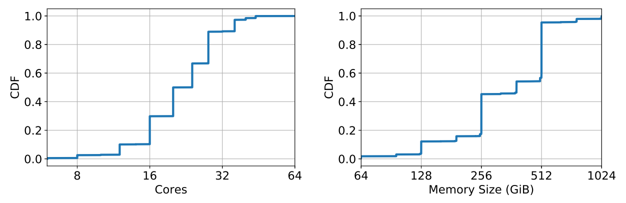
[2] We omit the platform identity in this submission to preserve the author anonymity.



(a) Nodes per Cluster **(b)** Consolidation Factor

Figure 1: Cluster-level Configuration

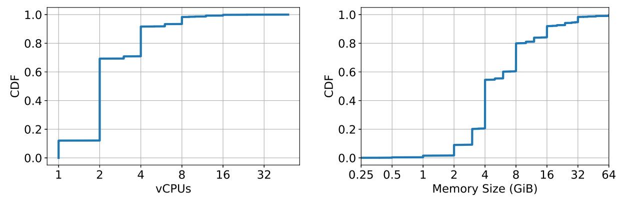
of the nodes have more than 24 physical cores and 384 GiB of RAM, and 10% have at least 36 cores and more than 512 GiB of RAM.



(a) CPU **(b)** Memory

Figure 2: Node-level Configuration

VM-level Configuration Figure 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.



(a) CPU **(b)** Memory

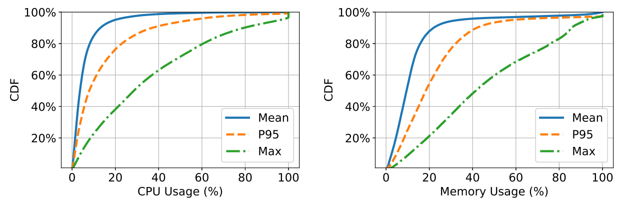
Figure 3: VM-level Configuration

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 SQLServer, MS Exchange, etc.

Summary: *Enterprise clusters are often small-sized single-tenant clusters, with powerful nodes, that support the workload requirements of small and medium-sized businesses.*

2.3 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 95th percentile (P95) of a series of 5-minute measurements for any given metric.



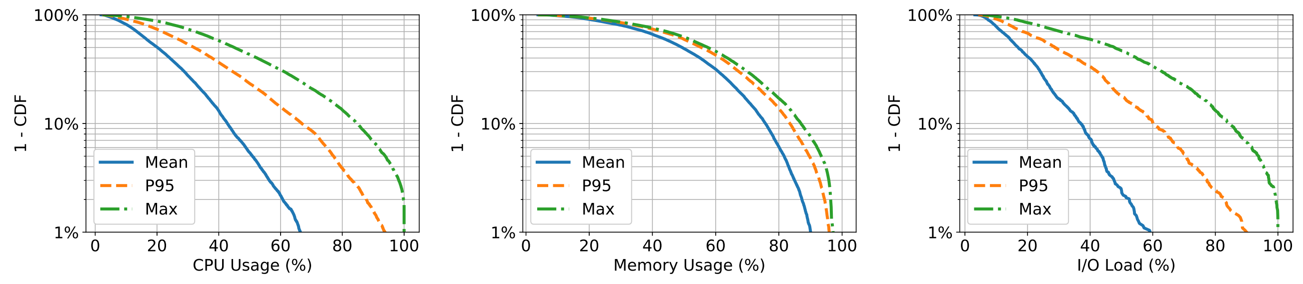
(a) CPU **(b)** Memory

Figure 4: VM Resource Usage

Figures 4a and 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.[3] Such allocated but sparsely used resources are the result of two main factors: (a) manual VM resource allocation, and (b) 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 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 I/O load. In general, we see that node usage is higher than

[3]Intuitively, the areas to the right of the maximum line in Figure 4a and 4b represent the global wasted resources that are never used, but our numbers additionally take into consideration the different absolute sizes of the VMs.

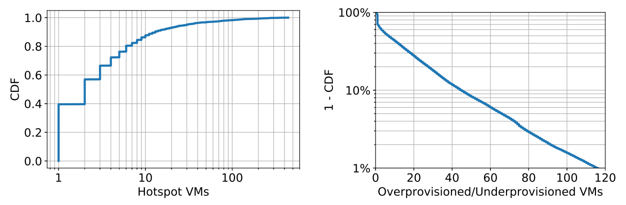


(a) CPU **(b)** Memory **(c)** I/O

Figure 5: Node Resource Usage

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 6a shows the distribution of hotspot VMs per cluster. We consider a VM to be a hotspot if its 95th 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

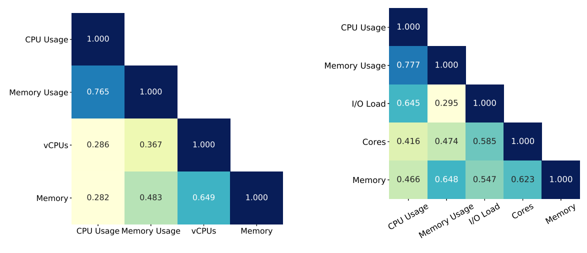
Figure 6: Hotspots and Over/Underprovisioned VMs Ratio

Summary: *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.*

2.4 Opportunities and Challenges for Adaptive Resource Allocation

Figure 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 [46], 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 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 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.

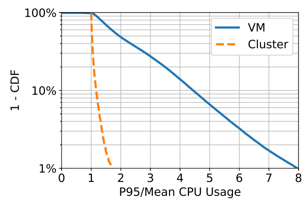


(a) VM-level **(b)** Node-level

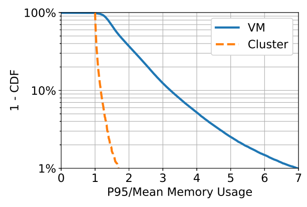
Figure 7: Provisioning and Utilization Metrics Correlations

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 8 shows the CCDF of the 95th percentile divided by the mean of CPU (8a) and memory (8b) usages for both VMs and clusters. We notice that ~45% of the VMs have a P95 at least 2x 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 8: P95/Mean Usage Ratios

Summary: *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.*

3 Design

This section describes the design of ADARES, a system that changes the physical resources allocated to VMs based on current workload and other attributes of the virtual execution environment. Our system crucially relies on the contextual bandits framework and other techniques 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. This section complements the system description with background information to assist readers unfamiliar with the contextual bandits framework.

3.1 ADARES 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 resource 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.

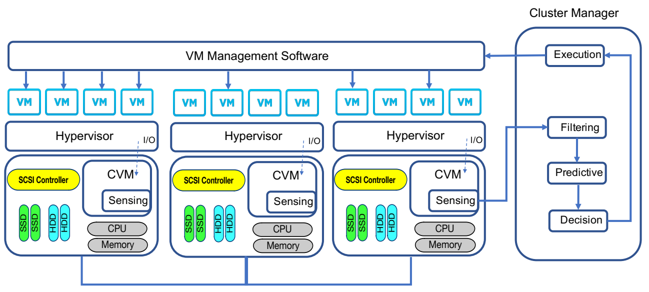


Figure 9: ADARES Architecture

Note that the VM assignment problem is orthogonal and is out of the scope of this paper, 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 vSphere/vMotion [45], 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 [43].

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 bandits 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.

3.2 ADARES Components

This section provides an overview of our system and introduce its core components. Figure 9 shows a high level overview its architecture. ADARES is composed of five core components to optimize VM configurations: the Sensing Service component is deployed on each node in the cluster, whereas the remaining components are executed within the cluster manager node.

3.2.1 Sensing Service (SS)

The Sensing Service is in charge of collecting telemetry data. The current version collections 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.

3.2.2 Filtering Service (FS)

The Filtering Service component serves as a pre-processing 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.

3.2.3 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 discuss these concepts in greater detail in §3.3.

PS exports two methods as part of its interface: (a) *predict*, which outputs the recommended actions for the selected VMs based on the ML model trained to maximize the expected reward, and (b) *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).

3.2.4 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: (a) *exploring* the configuration space to discover the rewards associated with a diverse set of actions, and (b) *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.

3.2.5 Execution Service (ES)

The decisions made by DS are handed to this service, which triggers the operations. Our current prototype supports integration with VMware vSphere,[4] which acts as the VM Management Software layer. Therefore, we also use VMware ESXi as the nodes' hypervisor. In order to perform provisioning changes on-the-fly, the underlying guest OS kernel needs support for hot addition/removal of CPUs and memory. However, VMware vSphere only provides native support for hot addition of both resources but not removal. We therefore use other vSphere APIs, in particular, the ability to execute programs directly on guests using the VMware tools installed on the VMs, to perform the adaptations. Finally, this component also keeps track of the execution progress and notifies the main controller of any failures during the process.

3.3 Contextual Multi-armed Bandit-based Approach

We now describe how ADARES uses contextual bandits for the VM resource allocation problem. We begin by describing the abstract contextual bandits framework and the rationale behind this 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] https://www.vmware.com/products/vsphere.html

3.3.1 Background

In the multi-armed bandit (MAB) problem with contextual information, an agent collects rewards for actions taken over a sequence of rounds. In each round, the agent chooses the action to take based on: (a) *context* (or features) of the current round, and (b) *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 *incomplete* [7].

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 $\mathbf{x}_t$, 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 $(\mathbf{x}_t, a_t, r_{t,a_t})$ [33].

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:

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

A fundamental challenge in bandit problems is the need for balancing exploration and exploitation. In order to minimize the regret in Equation 1, the agent *exploits* its past experience and chooses the action that appears to be the best. However, that action might be suboptimal due to the agent's insufficient knowledge. Instead, the agent may need to *explore* by selecting seemingly suboptimal actions in order to gather more knowledge about them [33]. Common applications of contextual bandits include, but are not limited to, personalized news recommendations, clinical trials, and mobile health interventions [50, 13].

3.3.2 Why Contextual Bandits?

Contextual bandits can be considered a hybrid 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 [13].

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 process of learning.

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. Reinforcement learning (RL) [49, 47, 48] 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? In our setting, however, we do not have 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. Further, although recent successes in deep reinforcement learning [36, 35] have made it quite popular among practitioners, most RL algorithms lack theoretical guarantees. On the contrary, there are many contextual bandit algorithms with strong theoretical guarantees that ensure convergence to an optimal solution [33, 7, 13], and they typically have a faster ramp up than their RL counterparts.

3.3.3 Contextual MAB Formulation for VM Resource Management

In order to apply contextual bandits to manage VM resources, we need to define the set of features that represent the contexts $\mathbf{x}$, 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.

We consider 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 MAB 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 before, 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 bandits 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 also 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.

3.3.4 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 interactions with the real VMs, which may be limited and costly [9, 29].

Inspired by the robotics community, herein, we build a cluster simulator to pre-train our agent. The idea is to then transfer the knowledge gained while training on this (costless) 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 component dependencies (e.g., hypervisors multiplexing shared resources), and the irregular interactions and 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 [25].

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 [6] 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. Although some initial results on the fidelity of our simulator are in Appendix A, 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.

3.4 Contextual Bandits meet ADARES

Having introduced the core constructs of ADARES, and MAB with contextual information , 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 §3.2 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 $\mathbf{x}_t \in \mathbb{R}^d$ 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 bandits 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). 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 this 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.

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 knwl)
 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
```

4 Evaluation

We implemented ADARES in about 7.8 KLOC of Python. Our current prototype is built in the context of the same 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.

4.1 Evaluation Setup

Clusters We have access to two experimental clusters. We have full control over one of the clusters, i.e., the smaller one, but we have limited access to the larger cluster. The first cluster is mainly homogeneous and consists of a total of 48 cores, a CPU capacity of 115.2 GHz and 512 GiB of RAM, on which we run ~20-36 VMs. The larger cluster has heterogeneous nodes and contains a total of 540 cores, 1.27 THz of CPU capacity, and 7.75 TiB of RAM. This latter cluster typically mimics real customer setups and is mainly used for regression tests and benchmarks of new releases. During our experiments, there were around ~530 VMs running, which our agents do not control.

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, which supports hot add/removal of CPU and memory. We use VMware vSphere APIs to execute programs on the guests to perform the adaptations. Only VMware Tools software needs to be installed in the guest OS, as the resources addition/removal can be done with native Linux programs (echo and grep) [2, 4]. Further, we clone the VMs in our experiments from the three instance types shown in Table 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, ± 1 for vCPUs and ± 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 2: VM Instance Types and their Min-Max Ranges

Workloads We simulate different workloads using a modified version of Flexible I/O Tester (FIO) [10], 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 our 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), and
- *bandits*, our method, where we adjust resources using contextual bandits.

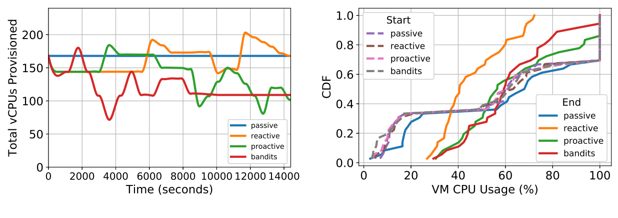
We use 75% as the underprovisioned threshold for the *reactive* and *proactive* baselines; that is, if the current or predicted VM resource usage (either CPU or memory) is above 75%, the system scales the resource(s) up. Similarly, we use a 25%-threshold to indicate overprovisioning, i.e., if the current or predicted VM resource usage is below that threshold, we scale the resource(s) down. 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 [15] with l_2 regularization and the squared loss, and we use the default hyperparameters of scikit-learn [39]. For our method, *bandits*, we use LinUCB [33], a popular Upper Confidence Bound (UCB) [5] algorithm. UCB algorithms are based on the principle of *optimism in the face of uncertainty*. On an incoming context, LinUCB computes the estimated reward and the uncertainty, and chooses the action with the highest score (estimated reward + uncertainty). We set the exploration constant to 0.5 (higher means more exploration), and the regularization parameter of the ridge regression to 0.01. Finally, our system makes decisions every 5 minutes.

4.2 Results

Static We start off by evaluating static workloads, which are characterized by a somewhat flat utilization profile over time. To that end, we run different static workload patterns across a set of 36 VMs, 12 of each of the instance types described in Table 2, in our controlled cluster, for a period of 4 hours. Figure 10a 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. Note that we only report results on the *bandits* version that uses transfer learning. The comparison of the *bandits* versions with and without transfer learning are in Appendix B.

Further, Figure 10b 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 ∼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%.

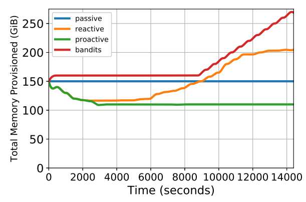
Overall, we see a 35% improvement, in terms of amount of vCPUs allocated, for the ML-based methods (*bandits* and *proactive*), when compared to static or threshold-based approaches. Further, at the end of the run, the standard deviation of the VMs CPU usage is 18% and 22% for *bandits* and *proactive* 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*.



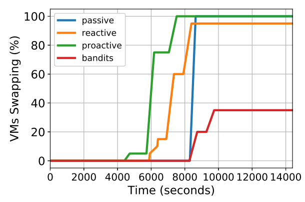
(a) Provisioned vCPUs **(b)** VM CPU Usage (Start-End)

Figure 10: Static Workload

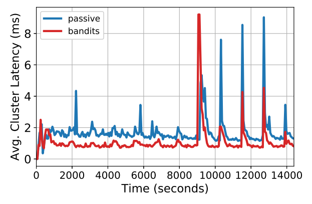
Increasing Another example of workloads we observe in practice are those with increasing resource demands. In this case, we simulate a workload with increasing working set size (WSS). We augment the WSS every 20 minutes for a group of 20 *xlarge* VMs running in our controlled cluster. Figure 11 shows the results of 4-hour runs. From 11a 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. 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 11b, 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 11c and 11d 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.



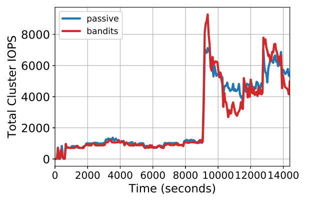
(a) Provisioned Memory



(b) VMs doing Swapping



(c) Cluster Latency

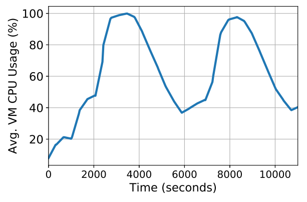


(d) Cluster IOPS

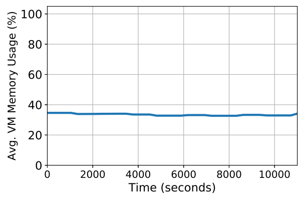
Figure 11: Increasing Memory Workload

Periodic and Static Finally, we focus on periodic and static workloads. In particular, we vary CPU utilization levels of VMs (Figure 12a), but keep constant the memory usage (Figure 12b). We 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 but without incurring in swapping.

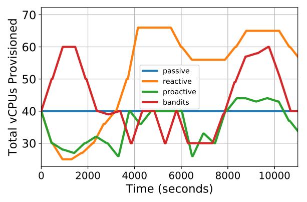
In this experiment, we deploy four AdaRes agents and execute them in parallel, one for each method, in the larger cluster. Each agent manages 10 *xlarge* VMs during the run. We select 5 nodes at random, and place 2 VMs of each agent in each one of the nodes in order not to benefit methods with VMs running on lightly loaded nodes. Recall that there are more than 500 VMs running in the cluster.



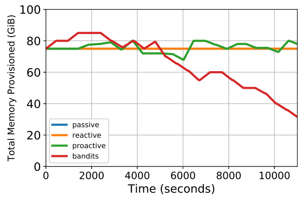
(a) CPU Pattern



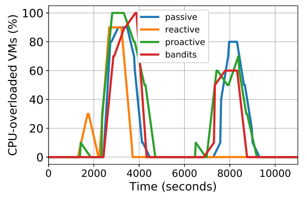
(b) Memory Pattern



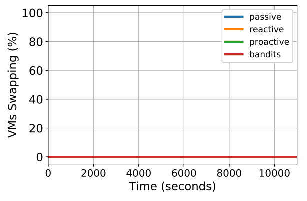
(c) Provisioned vCPUs



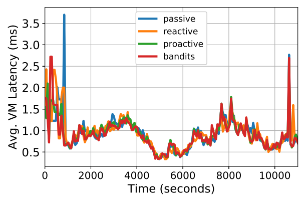
(d) Provisioned Memory



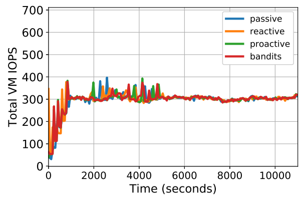
(e) VMs with CPU Overload



(f) VMs with Swapping



(g) VM Latency



(h) VM IOPS

Figure 12: Periodic and Static Workload

From Figure 12c we observe that *proactive* tends to provision less vCPUs than *reactive*, but with a similar pattern. On the other hand, *bandits* has a hard time at the beginning, but seems to be learning it should scale up during peak times. Or in other words, from Figure 12e we observe that the percentage of VMs with CPU overload during periods of high load decreases over time for the *bandits* method. Regarding memory, we observe that *bandits* is the only one that mainly scales down (12d), and it doesn't cause VMs to swap (12f). Last but not least, we see that VM latencies (12g) and IOPS

(12h) are comparable across methods.

Overall, *bandits* saves a total of 45 GiB of RAM, a 60% improvement over the other baselines, while at the same time keeps reducing the CPU overload during peak times. We speculate that longer runs would derail in more benefits to *bandits* (or any of the adaptive methods), as opposed to the *passive* one. Further, we acknowledge that different threshold settings can cause completely different behavior for the *reactive* and *proactive* approaches. Even for *bandits*, hyper-parameter 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.

5 Related Work

We discuss work relevant to AdaRes in the areas of measurements, resource management and ML for systems.

Measurements: Google traces [41, 56] have enabled research on a broad set of topics, from workload characterizations [34] to new algorithms for machine assignment [40]. 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 [18, 30], but mainly in the public cloud setting. Prior work on measurements of enterprise clusters [16] do not quantify issues related to VM resource allocations.

Resource Management: Auto-scaling systems, such as the ones offered in Google Cloud Platform [3] or Amazon Web Services [1], 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 [8].

Gmach et al. [26] propose a resource allocation system for datacenter applications that depends on predicting their behavior a priori based on the repetitive nature of their workloads. On a similar note, DejaVu [51] 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 an contextual bandits model to dynamically adapt to changes.

Ernest [52] is a system to efficiently run applications on shared infrastructure by choosing the right hardware configuration. 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.

A great deal of previous research into resource management has focused on VM/task scheduling and migration [37, 14, 58, 19, 43]. 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 [27].

ML for Systems: There is a significant body of work in the space of applying ML in order to optimize systems. With respect to the VM resource allocation problem, some of the most prominent work has been done by Delimitrou et al. [20, 21]. 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. Instead, in our work we use contextual bandits, which in some sense, encodes the notion of online profiling and re-adjustment using the exploration/exploitation trade-off.

PARIS [57] leverages an offline profiling framework and established machine learning techniques, such as random forests, to identify the best VM across multiple cloud providers. Some prior work also investigate horizontal autoscaling using threshold-based and RL techniques [23, 22]. They mainly focus on minimizing the cost of acquiring VMs in a public cloud setting and they assume all VMs are of equal size. Instead, we focus on vertical scaling of VMs of different sizes within enterprise clusters, using bandits, where we are concerned about resource efficiency in order to be able to cope with more (incoming) VM workloads.

6 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. 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 [44], 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 [31], which would allow the agent to have a better coverage of the state space, thus improving generalization and speeding up training.

7 Conclusions

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. As part of 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 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.

Acknowledgments

Special thanks to Kevin Jamieson and Lalit Jain for their valuable feedback on the contextual bandits formulation. We are also grateful to Robert Marver for helping with the experiments. One of the authors was supported by the Argentinean government program BEC.AR.

References

[1] Amazon Web Services Auto Scaling. https://aws.amazon.com/autoscaling/. Accessed: 2018-08-27.

[2] CPU hotplug in the Kernel. https://www.kernel.org/doc/html/v4.14/core-api/cpu_hotplug.html. Accessed: 2018-09-19.

[3] Google Cloud Platform AutoScaler. https://cloud.google.com/compute/docs/autoscaler/. Accessed: 2018-08-27.

[4] Memory Hotplug. https://github.com/torvalds/linux/blob/master/Documentation/memory-hotplug.txt. Accessed: 2018-09-19.

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

[6] Vdbench. https://www.oracle.com/technetwork/server-storage/vdbench-downloads-1901681.html. Accessed: 2018-08-28.

[7] AGARWAL, A., HSU, D., KALE, S., LANGFORD, J., LI, L., AND SCHAPIRE, R. E. 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.

[8] ARABNEJAD, H., PAHL, C., JAMSHIDI, P., AND ESTRADA, G. 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* (Piscataway, NJ, USA, 2017), CCGrid '17, IEEE Press, pp. 64–73.

[9] ARULKUMARAN, K., DEISENROTH, M. P., BRUNDAGE, M., AND BHARATH, A. A. A Brief Survey of Deep Reinforcement Learning. *CoRR abs/1708.05866* (2017).

[10] AXBOE, J. Flexible I/O Tester. https://github.com/axboe/fio, 2011.

[11] BANERJEE, I., GUO, F., TATI, K., AND VENKATASUBRAMANIAN, R. Memory Overcommitment in the ESX Server, 2011.

[12] BARKER, S. K., AND SHENOY, P. Empirical Evaluation of Latency-sensitive Application Performance in the Cloud. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems* (2010), MMSys '10, ACM, pp. 35–46.

[13] BEYGELZIMER, A., LANGFORD, J., LI, L., REYZIN, L., AND SCHAPIRE, R. Contextual bandit algorithms with supervised learning guarantees. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (Fort Lauderdale, FL, USA, 11–13 Apr 2011), G. Gordon, D. Dunson, and M. Dudk, Eds., vol. 15 of *Proceedings of Machine Learning Research*, PMLR, pp. 19–26.

[14] BOBROFF, N., KOCHUT, A., AND BEATY, K. A. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *Integrated Network Management* (2007), IEEE, pp. 119–128.

[15] BOTTOU, L. Large-scale machine learning with stochastic gradient descent. In *in COMPSTAT* (2010).

[16] CANO, I., AIYAR, S., AND KRISHNAMURHTY, A. Characterizing Private Clouds: A Large-Scale Empirical Analysis of Enterprise Clusters. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (2016), SoCC '16, ACM, pp. 29–41.

[17] CORTEZ, E., BONDE, A., MUZIO, A., RUSSINOVICH, M., FONTOURA, M., AND BIANCHINI, R. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles* (New York, NY, USA, 2017), SOSP '17, ACM, pp. 153–167.

[18] CORTEZ, E., BONDE, A., MUZIO, A., RUSSINOVICH, M., FONTOURA, M., AND BIANCHINI, R. 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).

[19] DELIMITROU, C., AND KOZYRAKIS, C. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2013), ASPLOS '13, ACM, pp. 77–88.

[20] DELIMITROU, C., AND KOZYRAKIS, C. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (2014), ASPLOS '14, ACM, pp. 127–144.

[21] DELIMITROU, C., AND KOZYRAKIS, C. 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* (2016), ASPLOS '16, ACM, pp. 473–488.

[22] DUTREILH, X., KIRGIZOV, S., MELEKHOVA, O., MALENFANT, J., RIVIERRE, N., AND TRUCK, I. Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: towards a fully automated workflow. In *7th International Conference on Autonomic and Autonomous Systems (ICAS'2011)* (Venice, Italy, May 2011), pp. 67–74.

[23] DUTREILH, X., MOREAU, A., MALENFANT, J., RIVIERRE, N., AND TRUCK, I. From Data Center Resource Allocation to Control Theory and Back. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing* (2010), CLOUD '10, IEEE Computer Society, pp. 410–417.

[24] FARLEY, B., JUELS, A., VARADARAJAN, V., RISTENPART, T., BOWERS, K. D., AND SWIFT, M. M. More for Your Money: Exploiting Performance Heterogeneity in Public Clouds. In *Proceedings of the Third ACM Symposium on Cloud Computing* (2012), SoCC '12, ACM, pp. 20:1–20:14.

[25] García, J., and Fernández, F. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.* (2015), 1437–1480.

[26] Gmach, D., Rolia, J., Cherkasova, L., and Kemper, A. Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization* (Washington, DC, USA, 2007), IISWC '07, IEEE Computer Society, pp. 171–180.

[27] Hermenier, F., Lawall, J., and Muller, G. BtrPlace: A Flexible Consolidation Manager for Highly Available Applications. *IEEE Trans. Dependable Secur. Comput. 10*, 5 (Sept. 2013), 273–286.

[28] Iosup, A., Yigitbasi, N., and Epema, D. On the Performance Variability of Production Cloud Services. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2011), CCGRID '11, IEEE Computer Society, pp. 104–113.

[29] Kansky, K., Silver, T., Mély, D. A., Eldawy, M., Lázaro-Gredilla, M., Lou, X., Dorfman, N., Sidor, S., Phoenix, D. S., and George, D. 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.

[30] Kilcioglu, C., Rao, J. M., Kannan, A., and McAfee, R. P. Usage Patterns and the Economics of the Public Cloud. In *Proceedings of the Twenty-Sixth International World Wide Web Conference* (2017).

[31] Krause, A., and Golovin, D. Submodular Function Maximization. In *Tractability*, L. Bordeaux, Y. Hamadi, and P. Kohli, Eds. Cambridge University Press, 2014, pp. 71–104.

[32] Langford, J., and Zhang, T. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds. Curran Associates, Inc., 2008, pp. 817–824.

[33] Li, L., Chu, W., Langford, J., and Schapire, R. E. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web* (New York, NY, USA, 2010), WWW '10, ACM, pp. 661–670.

[34] Mishra, A. K., Hellerstein, J. L., Cirne, W., and Das, C. R. Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters. *SIGMETRICS Perform. Eval. Rev. 37*, 4 (Mar. 2010), 34–41.

[35] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari With Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*. 2013.

[36] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level Control through Deep Reinforcement Learning. *Nature 518*, 7540 (02 2015), 529–533.

[37] Novaković, D., Vasić, N., Novaković, S., Kostić, D., and Bianchini, R. DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference* (Berkeley, CA, USA, 2013), USENIX ATC'13, USENIX Association, pp. 219–230.

[38] Pan, S. J., and Yang, Q. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng. 22*, 10 (Oct. 2010), 1345–1359.

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

[40] Reiss, C., Tumanov, A., Ganger, G. R., Katz, R. H., and Kozuch, M. A. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing* (New York, NY, USA, 2012), SoCC '12, ACM, pp. 7:1–7:13.

[41] Reiss, C., Wilkes, J., and Hellerstein, J. L. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, Nov. 2011.

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

[43] Ruprecht, A., Jones, D., Shiraev, D., Harmon, G., Spivak, M., Krebs, M., Baker-Harvey, M., and Sanderson, T. Vm live migration at scale. In *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (2018), VEE '18, ACM.

[44] Settles, B. Active Learning Literature Survey. Computer sciences technical report, University of Wisconsin–Madison, 2009.

[45] Setty, S. VMware vSphere 5.1 vMotion Architecture, Performance and Best Practices, 2012.

[46] Spearman, C. The proof and measurement of association between two things. *The American Journal of Psychology 15*, 1 (1904), 72–101.

[47] Sutton, R. S. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, 1984. AAI8410337.

[48] Sutton, R. S. Learning to Predict by the Methods of Temporal Differences. In *MACHINE LEARNING* (1988), Kluwer Academic Publishers, pp. 9–44.

[49] Sutton, R. S., and Barto, A. G. *Introduction to Reinforcement Learning*, 1st ed. MIT Press, 1998.

[50] Tewari, A., , and Murphy, S. A. From Ads to Interventions: Contextual Bandits in Mobile Health, 2017.

[51] Vasić, N., Novaković, D., Miučin, S., Kostić, D., and Bianchini, R. DejaVu: Accelerating Resource Allocation in Virtualized Environments. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems* (2012), ASPLOS XVII, ACM, pp. 423–436.

[52] Venkataraman, S., Yang, Z., Franklin, M., Recht, B., and Stoica, I. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (Santa Clara, CA, 2016), USENIX Association, pp. 363–378.

[53] VMware. Understanding Memory Resource Management in VMware ESX Server, 2011.

[54] VMware. Performance Best Practices for VMware vSphere 5.5, 2014.

[55] VMware. Performance Best Practices for VMware vSphere 6.0, 2015.

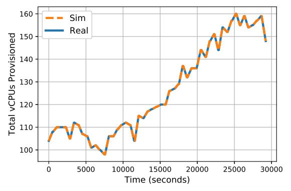
[56] Wilkes, J. More Google cluster data. Google research blog, Nov. 2011.

[57] Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., Smith, B., and Katz, R. H. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing* (2017), SoCC '17, ACM.

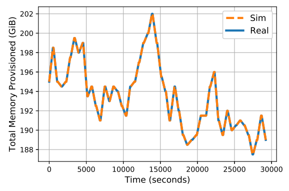
[58] Yang, H., Breslow, A., Mars, J., and Tang, L. Bubble-flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2013), ISCA '13, ACM, pp. 607–618.

A Cluster Simulator Fidelity

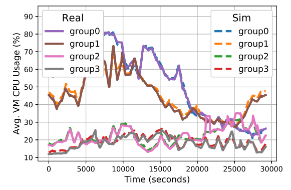
Herein, we evaluate the fidelity of our cluster simulator. We instantiate 26 *xlarge* VMs in our smaller cluster. We group them in four distinct groups, each with a different workload pattern and I/O intensity. We perform random configuration changes during a 8-hour period. We record all the actions done along the way, and we then replay those exact same actions in our simulator. Figures 13a and 13b 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 13c 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 13d, illustrates the memory usage across groups. Herein, 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.



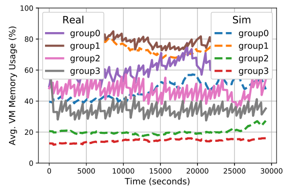
(a) Provisioned vCPUs



(b) Provisioned Memory



(c) CPU Usage



(d) Memory Usage

Figure 13: VM Resource Provisioning and Utilization

Finally, Figure 14 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.

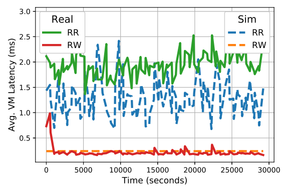
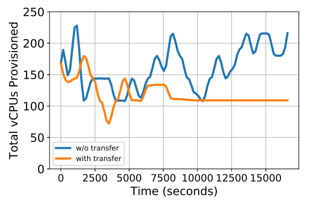


Figure 14: Latency

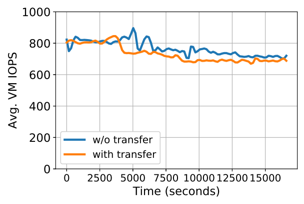
B Transfer Learning: *Sim2Real*

In this section we evaluate how transfer learning helps to speed up training in real clusters. We use the same setting as in 4.2, where we run different static workloads across a set of 36 VMs, 12 of each instance type, in our controlled cluster, 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 15a 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 2x 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 15b shows the average I/O operations per second of the VMs in this workload. We observe that, even though we saved 2x vCPUs, we are still able to perform very close to the vanilla bandit version in terms of IOPS.



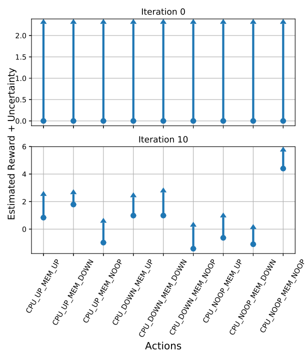
(a) Provisioned vCPUs



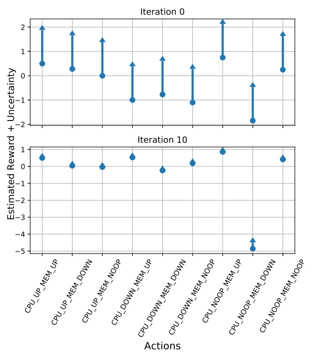
(b) VM IOPS

Figure 15: vCPUS Allocations and VM IOPS

We now illustrate how transfer learning helps LinUCB to accelerate training. Figure 16 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 16a). 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 16b 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 16: LinUCB and Transfer Learning

C LinUCB in Practice

Finally, we illustrate few more examples of how LinUCB operates in practice. We use our cluster simulator to replicate the smaller 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 17 and 18 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 17, 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 18 illustrates that scaling up both vCPUs and memory is the clear winner for VM contexts with underprovisioning of both resources.

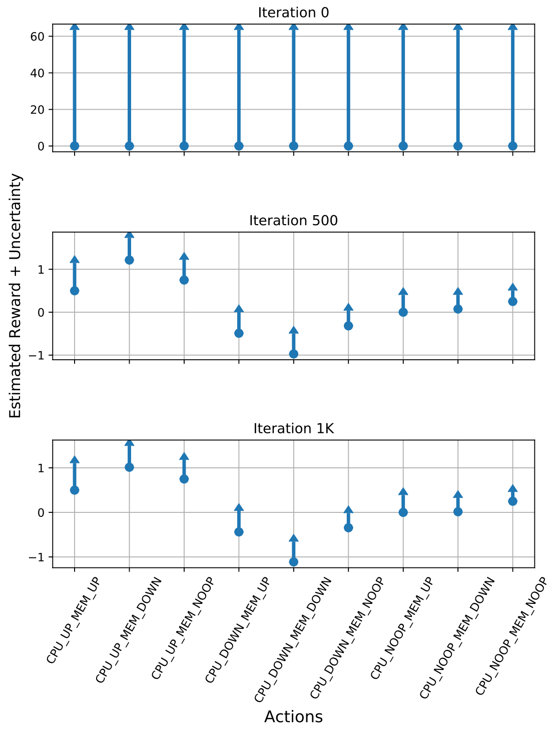


Figure 17: CPU Overload Context

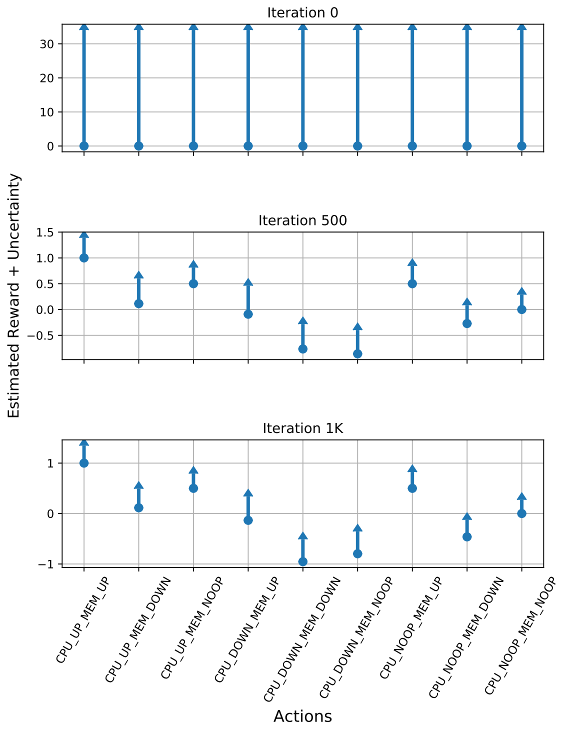


Figure 18: CPU Overload and Memory Swapping Context