Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud

Qi Zhang, *Student Member, IEEE*, Mohamed Faten Zhani, *Member, IEEE*, Raouf Boutaba, *Fellow, IEEE*, and Joseph L. Hellerstein, *Fellow, IEEE*

Abstract—Data centers consume tremendous amounts of energy in terms of power distribution and cooling. Dynamic capacity provisioning is a promising approach for reducing energy consumption by dynamically adjusting the number of active machines to match resource demands. However, despite extensive studies of the problem, existing solutions have not fully considered the heterogeneity of both workload and machine hardware found in production environments. In particular, production data centers often comprise heterogeneous machines with different capacities and energy consumption characteristics. Meanwhile, the production cloud workloads typically consist of diverse applications with different priorities, performance and resource requirements. Failure to consider the heterogeneity of both machines and workloads will lead to both sub-optimal energy-savings and long scheduling delays, due to incompatibility between workload requirements and the resources offered by the provisioned machines. To address this limitation, we present Harmony, a Heterogeneity-Aware dynamic capacity provisioning scheme for cloud data centers. Specifically, we first use the K-means clustering algorithm to divide workload into distinct task classes with similar characteristics in terms of resource and performance requirements. Then we present a technique that dynamically adjusting the number of machines to minimize total energy consumption and scheduling delay. Simulations using traces from a Google's compute cluster demonstrate Harmony can reduce energy by 28 percent compared to heterogeneity-oblivious solutions.

Index Terms—Cloud computing, workload characterization, energy management

1 Introduction

Data centers have recently gained significant popularity as a cost-effective platform for hosting large-scale service applications. While large data centers enjoy economies of scale by amortizing long-term capital investments over a large number of machines, they also incur tremendous energy costs in terms of power distribution and cooling. For instance, it has been reported that energy-related costs account for approximately 12 percent of overall data center expenditures [5]. For large companies like Google, a 3 percent reduction in energy cost can translate to over a million dollars in cost savings [22]. At the same time, governmental agencies continue to implement and regulations to promote energy-efficient computing [2]. As a result, reducing energy consumption has become a primary concern for today's data center operators.

In recent years, there has been extensive research on improving data center energy efficiency [24], [29]. One promising technique that has received significant attention is *dynamic capacity provisioning* (DCP). The goal of this technique is to dynamically adjust the number of active machines in a data center in order to reduce energy consumption while meeting the *service level objectives* (SLOs) of workloads. In the context of workload scheduling in data centers, a metric of particular importance is *scheduling delay* [20], [23], [25], [27], which is the time a request waits in the scheduling queue before it is scheduled on a machine. Task scheduling delay is a primary concern in data center environments for several reasons: (1) A user may need to immediately scale up an application to accommodate a surge in demand and hence requires the resource request to be satisfied as soon as possible. (2) Even for lower-priority requests (e.g., background applications), long scheduling delay can lead to starvation, which can significantly hurt the performance of these applications. In practice, however, there is often a tradeoff between energy savings and scheduling delay. Even though turning off a large number of machines can achieve high energy savings, at the same time, it reduces service capacity and hence leads to high scheduling delay. Finally, the heterogeneity-aware DCP scheme should also take into account the reconfiguration costs associated with switching on and off individual machines. This is because frequently turning on and off a machine can cause the "wear-and-tear" effect [12], [19] that reduces the machine lifetime.

Despite the fact that a large number of DCP schemes have been proposed in the literature in recent years, a key challenge that often has been overlooked or considered difficult to address is *heterogeneity*, which is prevalent in production cloud data centers [23]. We summarize the types of heterogeneity found in production environments as follows:

Machine heterogeneity. Production data centers often comprise several types of machines from multiple generations [25]. They have heterogeneous processor architectures and speeds, hardware features, memory and disk capacities. Consequently, they have different runtime energy consumption rates.

- *Q. Zhang, M.F. Zhani, and R. Boutaba are with David. R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada. E-mail: {q8zhang, mfzhani, rboutaba}@uwaterloo.ca.*
- *J.L. Hellerstein is with Google Inc., 651 34th St., Seatthe, WA 98103. E-mail: jlh@google.com.*

Manuscript received 2 Oct. 2013; revised 7 Jan. 2014; accepted 23 Jan. 2014; date of publication 14 Apr. 2014; date of current version 30 Apr. 2014. Recommended for acceptance by A. Wierman.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2306427

Workload heterogeneity. Production data centers receive a vast number of heterogeneous resource requests with diverse resource demands, durations, priorities and performance objectives [20], [25], [27]. In particular, it has been reported that the differences in resource demand and duration can span several orders of magnitude [8], [25], [27]. The heterogeneous nature of both machine and workload in production cloud environments has profound implications on the design of DCP schemes. In particular, given a rise of workload requests, a heterogeneity-oblivious DCP scheme can turn on wrong types of machines which are not capable of handling these requests (e.g., due to insufficient capacity), resulting in both resource wastage and high scheduling delays. However, designing a heterogeneity-aware DCP scheme can be difficult, because it requires an accurate characterization of both workload and machine heterogeneities. At the same time, it also requires a heterogeneity-aware performance model that balances the tradeoff between energy savings and scheduling delay at runtime.

In this paper, we present Harmony, a *H*eterogeneity-*A*ware *R*esource *MON*itoring and management s*Y*stem that is capable of performing DCP in heterogeneous data centers. Specifically, we first present a characterization of the heterogeneity found in one of Google's production compute clusters. Using standard K-means clustering, we show that the heterogeneous workload can be divided into multiple task classes with similar characteristics in terms of resource and performance objectives. We then formulate the DCP as an optimization problem that considers machine and workload heterogeneity as well as reconfiguration costs. We then devise online control algorithms based on the *Model Predictive Control* framework that solves the optimization problem at runtime. This paper is an extension of our previous work [28]. Specifically, we updated our runtime scheduling algorithm, provided a theoretical bound on the size of each task class to achieve an efficient tradeoff between scheduling delay and energy consumption, and evaluated the effect of resource over-provisioning on solution quality. Through experiments using traces from one of Google's compute clusters, we found Harmony achieves lower scheduling delay and energy consumption compared to heterogeneity-oblivious DCP solutions.

The rest of the paper is organized as follows. Section 2 surveys related work in the literature. Section 3 provides an analysis of a publicly available workload traces from Google to motivate our approach. Section 4 provides an overview of Harmony. In Section 5, we describe the way Harmony captures the runtime workload composition. We present the formulation of the heterogeneity-aware DCP in Section 6, and provide two technical solutions in Section 7. Section 8 discusses the deployment of Harmony in practice. Finally, we evaluate our proposed system using Google workload traces in Section 9, and draw our conclusions in Section 10.

2 Related Work

Characterizing workload in production clouds has received much attention in recent years, as both scheduler design and capacity upgrade require a careful understanding of the workload characteristics in terms of arrival rate, requirements, and duration [20]. For example, Mishra et al. have analyzed the workload of a Google compute cluster, and proposed an approach to task classification using k-means clustering [20]. Following the same line of research, Chen et al. provided a characterization of Google cluster workload at job-level applying the k-means algorithm [13]. Sharma et al. [25] and Zhang et al. [27] studied the problem of finding accurate workload characterizations through benchmark generation and validation. Recently, Reiss et al. [23] provided a comprehensive analysis of the heterogeneity and dynamism found in Google cluster traces, and found that both machine configurations and workload composition are highly heterogeneous and dynamic over time. They also pointed out the importance of considering workload heterogeneity for designing adaptive schedulers. However, the goal of these studies was to understand the workload composition in production clouds, rather than using workload characterization for resource allocation and capacity provisioning.

There is a large body of literature on energy-aware dynamic capacity provisioning in data centers. For example, pMapper [26] is a migration-aware workload placement framework for optimizing application performance and power consumption in data centers. However, it does not consider the cost of turning on and off machines when making provisioning decisions, nor does it consider task arrival rate and scheduling delay objectives. Similarly, Mistral [17] is a framework that dynamically adjusts VM placement to find a tradeoff between power consumption, application performance, and reconfiguration costs. However, it does not consider the arrival rate of task requests in its formulation. More recently, Ren et al. [24] studied the problem of scheduling heterogenous batch workload across geographically distributed data centers. Different from our work, they assume that the workload has already been divided into distinct types. They further assume that every task can be scheduled on any machine, which is not always the case as we shall demonstrate in Section 3. To the best of our knowledge, no previous work has applied task classification to dynamic capacity provisioning problem in heterogenous data centers. Thus, we design Harmony as a workload-aware DCP framework that can achieve both higher application performance and efficiency in terms of energy savings.

3 Workload Analysis

To understand the heterogeneity in production cloud data centers, we have conducted an analysis of workload traces for one of Google's production compute clusters [4][1] consisting of approximately $12{,}000$ machines. The workload traces contain scheduling events, resource demand and usage records for a total of $672{,}003$ jobs and $25{,}462{,}157$ tasks over a time span of 29 days. Specifically, a *job* is an application that consists of one or more tasks. Each *task* is scheduled on a single physical machine. When a job is submitted, the user can specify the maximum allowed resource *demand* for each task in terms of required CPU and memory size.

1. It should be mentioned that the same data set has been analyzed by Reiss et al. [23]. However, our analysis extends, and largely complements the results in [23].

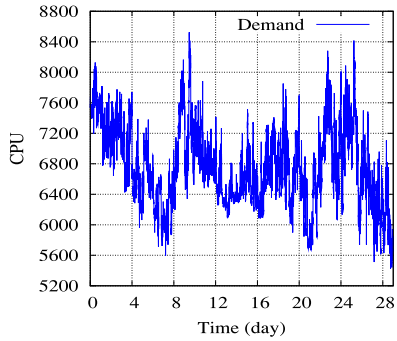


Fig. 1. Total CPU demand.

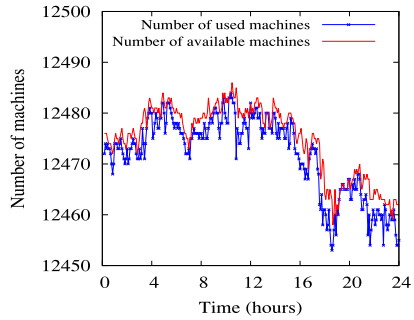


Fig. 3. Number of machines.

The values of the demand for each resource type were normalized between 0 and 1. Even though the data set does not provide task size for other resource types such as disk, it is straightforward to extend our approach to consider additional resource types.

In addition to resource demand, the user can also specify a scheduling class, a priority and placement constraints for each task. The *scheduling class* captures the type of the task. Its value ranges from 0 to 3, with 0 corresponding to least latency-sensitive tasks (e.g., batch processing tasks) and 3, the most latency-sensitive tasks (e.g., web servers). The scheduling class is used by every machine to determine the local resource allocation policy that should be applied to each task. The *priority* reflects the importance of each task. There are 12 priorities that are divided into three *priority groups*: *gratis*(0-1), *other*(2-8), *production*(9-11) [23]. Generally speaking, task priorities can be used for specifying the quality of service (QoS) in terms of desired task scheduling delay. During busy periods when demand approaches cluster capacity, task priorities can ensure that high priority tasks are scheduled earlier than low priority tasks, resulting in lower scheduling delay. In this work, we primarily analyze task characteristics at the priority group-level, because priority groups already provide a coarse-grained classification of tasks. In addition, they also have strong correlation with task scheduling classes [4], [23]. Nevertheless, our technical approach can be extended to handle any combination of task priority groups and task scheduling classes.

3.1 Machine and Workload Dynamicity

In our analysis, we first plot the total demand for both CPU and memory over time. The results are shown in Figs. 1 and 2, respectively. The total demand at a given time is determined by total resource requirement by all tasks in the system, including the tasks that are waiting to be scheduled. From both figures, it can be observed that the demand for each resource type can fluctuate significantly over time. Fig. 3 shows the number of machines available and used in the cluster. Specifically, a machine is available if it can be turned on to execute tasks, and is used if there is at least one task running on it. Fig. 3 also suggests that the capacity of the cluster is not adjusted according to resource demand, as the number of used machines is almost equal to the number of available machines. This suggests that a large number of machines can be turned off to save energy.

3.2 Analysis of Task Scheduling Delay

While turning off active machines can reduce total energy consumption, turning off too many machines can also hurt task performance in terms of scheduling delay. Fig. 4 shows the cumulative distribution function (CDF) of the scheduling delay for tasks with respect to their priority groups. It is apparent that tasks with production priority have better scheduling delay than the gratis ones. Indeed, more than 50 and 30 percent of the tasks in *production* and *other* priority groups respectively are scheduled immediately. On the other hand, some of the tasks were delayed significantly. During our analysis, we have also noticed that some tasks even with production priority were delayed for up to 21 days. Since the cluster is not constantly overloaded, the only possible explanation is that the task is difficult to schedule due to unrealistic resource requirement or had a placement constraint that is difficult to satisfy. These results suggest that more efficient provisioning and scheduling methods are needed to reduce the scheduling delay for these difficult-to-schedule tasks.

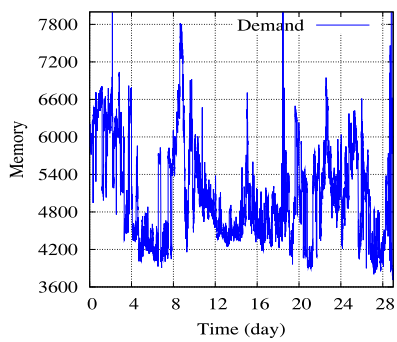


Fig. 2. Total memory demand.

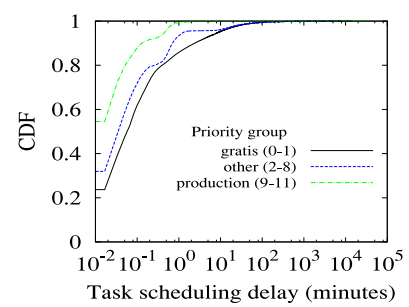


Fig. 4. CDF of task scheduling delay.

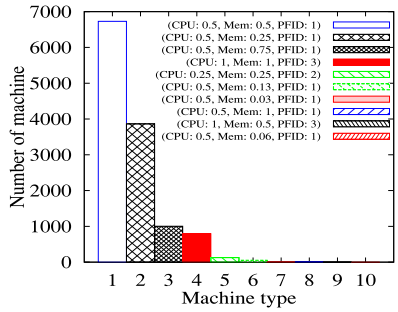


Fig. 5. Machine heterogeneity.

3.3 Understanding Machine Heterogeneity

The traces also provide information about the types of machines used in the cluster. A machine is characterized by its capacity in terms of CPU, memory and disk size as well as a platform ID, which identifies the micro-architecture (e.g., vendor name and chipset version) and memory technology (e.g., DDR or DDR2) of the machine. Similar to tasks, machine capacities are normalized such that the largest machine has a capacity equal to 1. Fig. 5 shows the different types of machines and their characteristics (capacity and platform ID (PFID)). We found 10 types of machines where more than 50 and 30 percent of the machines belong to machine types 1 and 2, respectively. On the other hand, machine types 3 and 4 have around 1,000 machines each. The remaining machine types (5 to 10) constitute less than 100 machines. Unfortunately, the traces do not provide detailed information about hardware specifications, however, it is likely that this heterogeneity translates into different energy consumption models.

3.4 Understanding Task Heterogeneity

In order to analyze the workload heterogeneity, we plotted tasks' requirements and their durations for the three priority groups. Fig. 7 shows the CPU and memory size of tasks belonging to each priority group. The coordinates of each point in these figures correspond to a combination of CPU and memory requirements. Radius of each circle is logarithmic in number of tasks within its proximity. It can be seen that most of the tasks have low resource requirements. In particular, we found that 43 percent of gratis tasks have the same CPU and memory requirements equal to 0.0125 and 0.0159, respectively. Furthermore, most of the large tasks are either CPU-intensive or memory-intensive. There is usually no correlation between CPU and

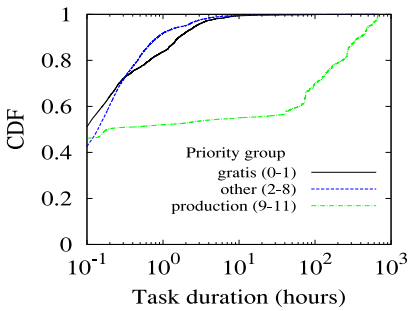


Fig. 6. CDF of task duration.

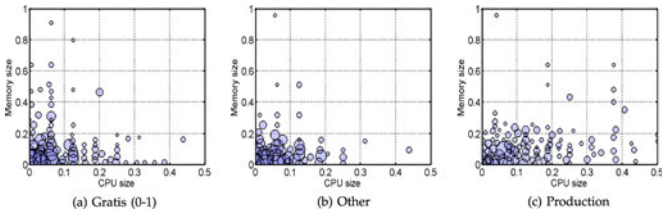


Fig. 7. Task size analysis.

memory requirements. Another key observation is that the difference in task size can span several orders of magnitude. For example, Fig. 7a shows that the largest task in the gratis priority group is almost 1,000× bigger than the smallest task in the same group for both CPU and memory. Similar characteristics can also be found in Figs. 7b and 7c. We note that similar characteristics have been observed in one of Facebook's data centers [15], suggesting these characteristics are likely to be common in Cloud data centers. Finally, from Figs. 5 and 7, it is clear that not every task (e.g., CPU size ≈ 1) can be scheduled on every type of machines (e.g., CPU capacity$= 0.5$).

Another important parameter that shows the heterogeneity of the tasks is the task duration. Fig. 6 shows the CDF of task durations for tasks with different priority groups. From Fig. 6, it can be seen that production tasks (9-11) have long durations that can reach up to 17 days, whereas 90 percent of the remaining tasks (i.e., gratis and other) have shorter duration that ranges between 0 and 10 hours. The same observation can be made for production-priority tasks when compared to other priority groups (Fig. 6). Furthermore, it is worth noting that more than 50 percent of the tasks are short (less than 100 seconds). This concurs with the previous studies [27], which showed that tasks are either short or long.

3.5 Summary

The above analysis suggests that while the benefit of dynamic capacity provisioning is apparent for production data center environments, designing an effective and dynamic capacity provisioning scheme is challenging, as it involves finding a satisfactory compromise between energy savings and scheduling delay with consideration to the heterogeneous characteristics of both machines and workload. In particular, we have found the heterogeneity in task size can span several orders of magnitude, and not every type of machine can schedule every task. Similar characteristics have also been recently reported in Microsoft and Facebook data centers [8]. Thus, it is a critical issue to design heterogeneity-aware DCP schemes for production data centers, as failing to consider these heterogeneous characteristics will result in sub-optimal performance.

4 System Overview

As discussed previously, we design Harmony as a DCP framework that considers both task and machine heterogeneity. This requires (1) an accurate characterization of both workload and machines, (2) effectively capture the dynamic workload composition at runtime, and (3) using the captured information to control the number of machines in the compute cluster to achieve a balance

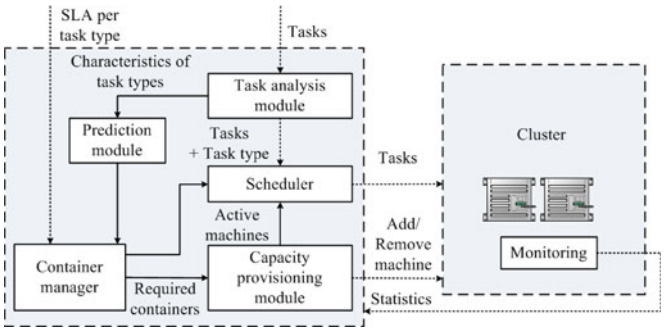


Fig. 8. System architecture.

between energy savings and scheduling delay. In practice, large cloud infrastructures such as Google compute clusters execute millions of tasks per day. Capturing heterogeneity at fine-grained (i.e., per-task) level is not a viable option due to the high overhead of monitoring and computation. Thus, a medium-grained characterization of the workload is necessary. To this end, we present a workload characterization of Google traces by dividing tasks into *task classes* using the K-means algorithm. However, different from previous work [13], [20] whose main objective is to understand workload characteristics, our goal is to find accurate workoad characterization, while supporting task classification (e.g., labeling) at runtime. Note that machines are naturally characterized (i.e., there are 10 types of machines in the cluster). Thus, our solution will mainly focus on task characterization.

Once the workload characterization has been obtained, we introduce a monitoring mechanism that allows Harmony to capture the runtime workload composition in terms of arrival rate for each task class. To make provisioning decisions, we define a *container* as a logical reservation of resources that is meant to host tasks belonging to the same task class. In our approach, the task containers serve as reservations for helping the controller to make machine allocation decisions (described in Section 7.2). It is also possible to directly use task containers for scheduling (described in Section 7.3). Finally, a heterogeneity-aware DCP controller is designed to adjust the number of active machines, based on the current machine availability and workload composition.

The architecture of Harmony is shown in Fig. 8. It consists of the following components. *The task analysis module* is responsible for monitoring the arrival of every task in order to identify the type to which it belongs. *The scheduler* is responsible for assigning incoming tasks to active machines in the cluster. *The prediction module* receives statistics of the arrival rate for each task class, and forecasts its future arrival rates. *The container manager* evaluates the number of containers required to schedule the current workload based on two parameters: (1) the predicted arrival rate, and (2) the required average scheduling delay for each type of tasks. The container manager periodically notifies the capacity provisioning module about the number of required containers for each type of tasks. *The capacity provisioning module* decides which machine in particular should be switched on or off. Obviously, the goal is to select the right combination of machines that can host the containers and, at the same time, minimizes the energy consumption. Finally, *The monitoring module* is responsible for collecting diverse statistics about tasks and machines, including CPU and memory usage, free resources and current task durations. It also reports any failures and anomalies to the management framework. In the following sections, we describe the design of Harmony in details.

5 Workload Analysis and Modeling

5.1 Task Classification

The goals of task classification is to divide tasks into classes with similar resource demand and performance characteristics. For the purpose of resource provisioning, it is necessary to consider task priority group, task size (CPU, memory) as well as task running time as the features for clustering. Specifically, the size of a task i can be modeled as a vector $s^i = (s^{i1}, \ldots, s^{iF})$, where F denotes the set of features used for clustering. Let N_k denote the tasks that belong to cluster k. Then, the centroid of each cluster can be defined as a vector $\overline{\mu}^k = (\overline{\mu}^{k1}, \ldots, \overline{\mu}^{kF})$, where $\overline{\mu}^{kr} = \frac{1}{|N_k|} \sum_{s^i \in N_k} s^{ir}$. The K-means clustering algorithm essentially tries to minimize the following similarity score:

$$score = \sum_{i=1}^{K} \sum_{i \in N_k} \|s^i - \overline{\mu}^k\|^2,$$

where $\|a - b\|$ denotes the Euclidian distance between two points a and b in the feature space. Even though Harmony does not restrict the type of clustering algorithm used for clustering, in practice we found K-means is simple and sufficient to serve our purpose.

A key issue associated with the use of K-means clustering is to determine the value of K, which is the number of clusters to be produced by the algorithm. A small value of K will lead to low-quality workload characterizations, which reduces the benefit of heterogeneity-aware DCP. On the other hand, a large value of K will lead to high monitoring and management overhead. In our scheme, we adopt a common approach which is to pick the value of K such that adding another cluster does not achieve much better gain in terms of minimizing $score$. We shall report the result of running the K-means clustering algorithm in Section 9.1.

Once a characterization of the workload has been made, the next challenge to be addressed is runtime task classification. Specifically, when a task arrives, Harmony needs to determine which task class (i.e., one of the K clusters) it belongs to. An easy solution is to compute the euclidean distance between the task and each of the centroids, and assign the task to the class that has the shortest euclidean distance. However, this cannot be done directly at runtime. This is because even though the resource requirements are known, the task running time is generally unknown to the system until the task finishes. In Harmony, this issue is addressed by leveraging the fact that tasks are either short or long, and the majority of the tasks are short tasks. Thus, we can initially label all tasks as short tasks, and gradually update the labels to the correct ones as time passes. Since only a small fraction of tasks are long, the error caused by the incorrect labeling is both small and short-lived.

Based on the above observation, we adopt a two-step approach for workload clustering. In the first step, tasks are classified based on static characteristics (e.g., CPU and memory size specified in the job request) using the K-means algorithm. In the second step, each task class is further divided into short tasks and long tasks using K-means algorithm. This two-step approach is reasonable because task running-time does not simply correlate with resources allocated on the hosting machine [7]. At runtime, each task is initially assumed to be short. Later on if the task running time exceeds the partitioning threshold between short and long tasks determined by the K-means algorithm, the task will be relabelled and assigned to the right task class. This clustering procedure reduces the error introduced by the labeling process.

5.2 Resource Prediction

Once the incoming tasks are classified, the prediction module is responsible for forecasting the arrival rate of each task class. Currently, we have implemented a time series-based predictor using the ARIMA [9] model, which has been shown to be effective for predicting workload arrival rate [29]. However, Harmony can adopt other demand prediction models as well.

Once the predicted task arrival rates have been obtained, the next step is to determine the combination of machines that need to be provisioned in next control period. In Harmony, the container manager is responsible for computing the number of containers required to support the workload of each task class. Specifically, let c_i denote the number of containers for tasks type i such that the average scheduling delay is equal to $\overline{d_i}$. We can model the queue of tasks of type i and its corresponding N_t^i containers at time t by $M/G/N_t^i$ queue since a single container can process one task at a time. Based on queuing theory, the average waiting time d_i for type i tasks is given by [16]:

$$d_i \approx \frac{\pi_{N_t^i}}{1-\rho_i} \cdot \frac{1+CV_i^2}{2} \cdot \frac{1}{N_t^i \mu_i}, \tag{1}$$

where μ_i is the execution rate of task type i, $\rho_i = \frac{\lambda_i}{N_t^i \mu_i}$ is the traffic intensity of tasks type i, CV_i^2 is the squared coefficient of variation of the average duration, and $\pi_{N_t^i}$ is the probability that a task has to wait in the queue:

$$\pi_{N_t^i} = \frac{(N_t^i \rho)^{N_t^i}}{N_t^i!(1-\rho_i)} \left[\sum_{k=0}^{N_t^i-1} \frac{(N_t^i \rho_i)^k}{k!} + \frac{(N_t^i \rho_i)^{N_t^i}}{N_t^i!(1-\rho_i)} \right]^{-1}. \tag{2}$$

Given an average scheduling delay and using Eq. (1), it is easy to estimate N_t^i to ensure $d_i \leq \overline{d_i}$ and $\rho_i < 1$.

In our experiments, we found this queuing model generally works well for estimating task resource requirements except for long-running tasks, for which queuing theory makes inaccurate resource predictions. We found a simple solution to address this limitation is to estimate the number of long running tasks using the ARIMA model. As each task runs for a very long time, the number of required containers is practically the number of long running tasks. As a result, we use the ARIMA model to predict the number of long running tasks, which translates into the number of required containers.

TABLE 1
Table of Notations

Symbol	Meaning
d_i	Average scheduling delay for task class i
R	Resource types
s^{kr}	Size of task i for resource type r
c^{kr}	Size of a container of type i for resource type r
C^{mr}	Capacity of a type m machine resource type r
$E^{idle,m}$	Energy consumption of a type m machine when idle
$\alpha^{\overline{m}r}$	Energy efficiency ratio of a type m machine for type r
u_t^{ir}	Util. of machine i for resource type r at time t
y_t^i	Boolean var. indicating machine i is active at time t
υ_t^i	Change in machine i's state at time t
a_t^{ik}	Num. of type k containers in machine i at time t
γ_t^{ik}	Change in a_t^{ik} at time t
z_t^{ik}	Number of type m machines active at time t
δ_t^m	Change in the num. of type m machines at time t
x_t^{mk}	Num. of type n containers in machine m at time t
σ_t^{mk}	Change in x_t^{mk} at t

6 The Capacity Provisioning Problem

We now provide a formal model for DCP in heterogenous environments, In our model, time is divided into intervals of equal duration, and control decision is made at the beginning of each time interval. The cluster consists of M types of machines. Let N_t^m denote the set of type m machines available (either active or not) at time interval t. Denote by $C^{mr} \in \mathbb{R}^+$ the capacity of a single machine of type $m \in M$ for resource type $r \in R$. Similarly, there are K types of containers to be scheduled at time t, the number of containers of type k is N_t^k. Let $c^{kr} \in \mathbb{R}^+$ denote the size of a type k container for resource type $r \in R$.

Let $y_t^i \in \{0,1\}$ denote whether machine i is active at time t. Define $\upsilon_t^i \in \{-1,0,1\}$ as an integer variable that indicates whether the machine is turned on ($u_t^i = 1$) or off ($u_t^i = -1$), or unchanged ($u_t^i = 0$). Also, let $a_t^{ik} \in \mathbb{N} \cup \{0\}$ as an integer variable that indicates the number of type n containers on machine i at time t, and γ_t^{ik} as the change in a_t^{ik} at time t. We thus have the following state equations:

$$y_{t+1}^i = y_t^i + \upsilon_t^i, \tag{3}$$

$$a_{t+1}^{ik} = a_t^{ik} + \gamma_t^{ik}. \tag{4}$$

Here, y_t^i and a_t^{ik} are variables that capture the state of the system at time t, whereas υ_t^i and γ_t^{ik} are the actual decision variables that need to controlled by the capacity provisioning module. The utilization of type r resource on machine i at time t can be computed as:

$$u_t^{ir} = \frac{1}{C^{mr}} \sum_{r \in R} a_t^{ik} c^{kr}. \tag{5}$$

As total energy usage of a physical machine can be estimated by a linear function of resource utilization [29], the energy consumption of all the active machines at time t can be computed as:

$$E_t = p_t \sum_{m \in M} \sum_{i \in N_t^m} y_t^i \left(E^{idle,m} + \sum_{r \in R} \alpha^{mr} u_t^{ir} \right), \tag{6}$$

where $E^{idle,m} \in \mathbb{R}^+$ is the energy consumption of a type m machine when it is idle, and $\alpha^{mr} \in \mathbb{R}^+$ is the slope of the energy consumption function. We can define $E_t^{idle} = p_t \sum_{m\in M} \sum_{i\in N_t^m} y_t^i E^{idle,m}$, $E_t^{util} = p_t \sum_{m\in M} \sum_{i\in N_t^m} \sum_{r\in R} \alpha^{mr} u_t^{ir}$ and rewrite E_t as $E_t = E_t^{idle} + E_t^{util}$.

To model task scheduling delay, since it is not possible for all containers to be scheduled when demand exceeds data center capacity, we assume there is a utility function $f^k(\cdot)$ that models the monetary gain for scheduling containers. $f^k(\cdot)$ is assumed to be a concave function that can be derived from SLO objectives. For example, $f^k(a^k)$ can be a linear decreasing function of the average scheduling delay computed by equation (1). The total performance utility can now be computed as:

$$U_t^{perf} = \sum_{k\in K} f^k\left(\sum_{m\in M}\sum_{i\in N_t^m} a_t^{ik}\right). \tag{7}$$

The machine switching cost can be described by:

$$C_t^{sw}(\upsilon_t^i) = \sum_{m\in M}\sum_{i\in N_t^m} q^{on,m}(\upsilon_t^i)^+ + q^{off,m}(\upsilon_t^i)^-, \tag{8}$$

where $q^{on,m} \in \mathbb{R}^+$ and $q^{off,m} \in \mathbb{R}^+$ denotes the cost for turning on and off of a single type m machine, respectively. Finally, Equation (9) ensures that containers scheduled on the same machine do not exceed the resource capacity of the machine:

$$\sum_{k\in K} a_t^{ik} c^{kr} \le y_t^i C^{mr} \quad \forall m \in M, i \in N_t^m, t \in \mathcal{T}. \tag{9}$$

Thus, the overall objective of DCP is to control the number of active machines and to adjust container placement in a way that maximizes the total performance gain in terms of scheduling delay, while minimizes the energy consumption and machine switching cost over a time horizon $\mathcal{T} = \{0, 1, \ldots, T\}$:

$$\max_{a_t^{ik}, \upsilon_t^i, y_t^i, \gamma_t^{ik}} R_T = \sum_{t=0}^{T} U_t^{pref} - E_t - C_t^{sw} \tag{DCP}$$

subjects to constraints (3), (4), and (9).

DCP is $\mathcal{NP}$-hard to solve as it generalizes the vector bin-packing problem [11]. Furthermore, linear programming based solutions cannot be applied to DCP due to the large number of variables involved. For example, given 10 task classes and over $10K$ machines, DCP contains at least $100K$ variables, making it difficult to solve in online settings. Finally, traditional bin-packing heuristics (e.g., First-Fit (FF)) do not apply directly to DCP as they do not consider machine switching costs.

7 Solution Techniques

Realizing that directly solving DCP is not possible, in this section we present two fast heuristics for solving DCP. Both techniques rely on solving the integer-relaxation of DCP (i.e., relaxing the constraints that variables must take integer values) called $DCP - RELAX$, which is much easier to solve than DCP. Once the solution for $DCP - RELAX$ is obtained, one of our solution techniques called container-based provisioning (CBP) directly rounds the numbers of machines to the nearest integer values and use these values for capacity provisioning. On the other hand, the container-based scheduling (CBS) technique attempts to find a feasible placement of containers in physical machines, and use containers for run-task scheduling. In both cases, the capacity provisioning module first adjusts the number of active machines, and informs the scheduler about how tasks should be assigned to each type of machines. In this section, we shall first present the formulation of $DCP - RELAX$, followed by a description of CBP and CBS in details. The benefits and limitations of each approach will be also be discussed in Section 8.

7.1 The Relaxation of DCP

In $DCP - RELAX$ we relax the integer constraints so that the number of machines (i.e., y_{it}) and container assignment (i.e., z_t^{ik}) can take real values. This relaxation yields a simpler formulation, as we only need to solve the total number of containers for each type of machines, rather than solving the number of containers per machine. Specifically, we denote by $z_t^m \in \mathbb{R}^+$ the number of type m machines that are active at time t, and $\delta_t^m \in \mathbb{R}$ the change in the number of active machines at time t. Similarly, define $x_t^{mk} \in \mathbb{R}^+$ as the number of type k containers assigned to machines of type m that is capable of hosting containers of type k, and $\sigma_t^{mk} \in \mathbb{R}^+$ the change in x_t^{mk} at time t. We thus have the following state equations:

$$z_{t+1}^m = z_t^m + \delta_t^m, \tag{10}$$

$$x_{t+1}^{mk} = x_t^{mk} + \sigma_t^{mn}. \tag{11}$$

As x_t^{mk} can take fractional values, we need to ensure that containers of each type can only be assigned to machines that are capable of hosting them. This is achieved by introducing a predefined boolean variable ψ^{mk} that indicates whether a container of type n can be scheduled on a machine of type m. We thus have the following schedulability constraint:

$$c_{kr} x_t^{mn} \le z_t^m \psi^{mk} C^{mr} \quad \forall m \in M, k \in K, r \in R, t \in \mathcal{T}. \tag{12}$$

$DCP - RELAX$ can now be stated as:

$$\max_{\delta_t^m, \sigma_t^{mk}} \sum_{t=0}^{T}\sum_{m\in M} -p_t\left(z_t^m E^{idle,m} + \sum_{r\in R}\sum_{k\in K} \frac{\alpha^{mr} c^{kr}}{c^{mr}} \cdot x_t^{mk}\right) + \sum_{t=0}^{T}\sum_{k\in K} f^k\left(\sum_{m\in M} x_t^{mk}\right) - \sum_{m\in M} C_t^{sw}(\delta_t^m), \tag{DCP-RELAX}$$

$$\begin{aligned} \text{subject to } & z_t^m \le N_t^m \qquad \forall m \in M, t \in \mathcal{T} \\ & x_t^{mk}, z_t^m \in \mathbb{R}^+ \qquad \forall k \in K, m \in M, t \in \mathcal{T}, \end{aligned} \tag{13}$$

along with constraints (10), (11) and (12). This problem is a convex optimization problem that can be solved using standard methods [10].

7.2 Container-Based Provisioning for DCP

Container-based provisioning is a simple heuristic for solving DCP. After $DCP - RELAX$ is solved, CBP simply rounds up the fractional values of (x_t^{mk}, z_t^m) to obtain an

integer solution for DCP, which gives the number of machines to be provisioned (i.e., $\lceil z_t^m \rceil$) and the number of type n tasks that should be scheduled on type m machines (i.e., $\lceil x_t^{mk} \rceil$). However, at runtime, the scheduler needs to ensure that the number of type n tasks assigned to type m machines must respect the provisioned capacity $\lceil x_t^{mk} \rceil$. A simple strategy is to ensure the number of type k tasks assigned to m (denoted by $Assign_t^{mk}$) is proportional to the number of containers:

$$Assign_t^{mk} = \frac{x_t^{mk}}{\sum_{j \in M} x_t^{jk}}.$$

This can be achieved easily by using a weighted round-robin scheduling policy. Furthermore, it can be easily integrated with existing scheduling algorithms. For example, variants of first-fit and best-fit algorithms (which are used in production clouds such as Microsoft [18], Google [25] and Open source platforms such as Eucalyptus [3]) can adopt this mechanism by changing the scheduling policy to weight round-robin first fit and weight round-robin best fit, respectively.

A key drawback of the above rounding scheme is that it often under-estimates the required capacity. The reason is that the fractional solution of $DCP - RELAX$ assumes that each container can be arbitrarily divided and placed on multiple machines. However, in practice, this is not realizable because each container must be scheduled on a single machine. Realizing that $DCP - RELAX$ under-estimates the required machines' capacities, we define an over-provisioning factor $\omega^k \in \mathbb{R}^+$ for each container type k, which captures the extra resource required to fully pack the type n containers. To account for ω^k, it suffices to replace equation (12) by the following equation:

$$\sum_{k \in K} \omega^k c^{kr} x_t^{mk} \leq z_t^m C^{mr} \quad \forall m \in M, r \in R, t \in \mathcal{T}. \tag{14}$$

The value of ω^k can be obtained through experiments. For example, we have found that setting $\omega^k = 1.2$ for all $n \in N$ seems to be a reasonable value in practice.

The main benefit of CBP is its simplicity and practicality for deployment in existing systems. However, the main drawback of CBP is that it still relies on bin-packing algorithms for scheduling. At runtime, tasks of different classes can still compete for resources in each type of machine. As a result, CBP does not provide high performance guarantee in terms of task scheduling delay.

7.3 Container-Based Scheduling for DCP

In this section, we present an alternative solution to CBP called *container-based scheduling*. Unlike CBP that uses bin-packing algorithms for scheduling, CBS allocates containers in each physical machine and use them for runtime task scheduling. Specifically, a type k container represents a resource reservation for tasks of type k. The number of type k containers on a machine i indicates the number of type k tasks that can be scheduled on machine i. At runtime, CBS adopts the following simple scheduling policy: each task is scheduled in the first available container such that scheduling the task on the machine does not cause machine capacity violation. If none of the machines can schedule the task without violating machine capacity constraint, the task will be kept in the scheduling queue.

The main benefit of CBS is that it provides low scheduling delay due to resource reservations on each machine. However, it also introduces several challenges which we shall discuss in the following sections.

7.3.1 Modeling Container Size

One of the main challenges for container-based scheduling is to select appropriate container size. Unlike in CBP where we can simply use the centroid to determine the container size, in CBS we need to set the container size large enough to ensure that with high probability, each task can be scheduled without exceeding the capacity of the physical machine. Specifically, setting the container size equal to the maximum possible container size can cause resource wastage due to over-estimation of true task resource demand. In contrast, setting the container size equal to the average task size will lead to under-estimation of task resource usage, resulting in tasks unschedulable in machines with available containers.

To address this issue, we rely on the statistical multiplexing of task resource demand to ensure the probability of machine capacity violation is low. Specifically, the result of the K-means clustering algorithm divides the feature space into K partitions, where every point in the space belongs to exactly one partition (i.e., the partition whose centroid has shortest distance to the point). We assume the tasks in each partition $1 \leq k \leq K$ are independently distributed according to a common distribution $\mathcal{D}^k$ (which can be an arbitrary distribution) with mean $\mu^k = (\mu^{k1}, \ldots, \mu^{kR})$ and standard deviation $\sigma^k = (\sigma^{k1}, \ldots, \sigma^{kR})$. Our goal is to select the container size $c^k = (c^{k1}, \ldots, c^{iR})$ for each task class $1 \leq k \leq K$ to ensure that given a task j of type k to be scheduled, the probability a task cannot be scheduled on any of the machines that have available type k containers is less than a small value ϵ . Mathematically, given M^k machines with available type k containers, let N^n denote the tasks scheduled on each machine $n \in M^k$. Given a task i to be scheduled, we want to ensure that

$$\prod_{m \in M^k} \Pr\left(\exists r : \sum_{j \in N} s^{jr} + s^{ir} > C^{mr} \middle| \sum_{j \in N} c^{jr} + c^{ir} \leq C^{mr} \right) \leq \epsilon. \tag{15}$$

Theorem 1. *Assume each task s^{kr} in each class k is independently and identically distributed with mean μ^{kr} and standard deviation σ^{kr} for each resource type r. Also, let M^k denote the minimum number of machines on which a type k task can be scheduled. We can set container size of task type k to*

$$c^{kr} = \mu^{kr} + \sqrt{\frac{|R| - \epsilon^{\frac{1}{M^k}}}{\epsilon^{\frac{1}{M^k}}}} \sigma^{kr}$$

for each $r \in R$ to ensure equation (15) holds.

Proof. Define $\overline{N} = N \cup \{s^{jr}\}$. Since

$$\begin{aligned}
&\Pr\Big(\exists r : \sum_{j\in N} s^{jr} + s^{ir} > C^{mr} \Big| \sum_{j\in N} c^{jr} + c^{ir} \le C^{mr}\Big) \\
&= \Pr\Big(\exists r : \sum_{j\in \overline{N}} s^{jr} > C^{mr} \Big| \sum_{j\in \overline{N}} c^{jr} \le C^{mr}\Big) \\
&\le \Pr\Big(\exists r : \sum_{j\in \overline{N}} s^{jr} > \sum_{j\in \overline{N}} c^{jr}\Big),
\end{aligned}$$

given M^k machines that have containers available, if we can ensure that the probability of violating machine capacity constraint is less than $\epsilon^{\frac{1}{M^k}}$ (i.e., $\Pr(\exists r : \sum_{j\in\overline{N}} s^{jr} > \sum_{j\in\overline{N}} c^{jr}) \le \epsilon^{\frac{1}{M^k}}$, then the inequality will hold. Furthermore, since $\Pr(\exists r : \sum_{j\in\overline{N}} s^{jr} > \sum_{j\in\overline{N}} c^{jr}) \le \sum_{r\in R} \Pr(\sum_{j\in\overline{N}} s^{jr} > \sum_{j\in\overline{N}} c^{jr})$, the bound will hold if we can ensure $\Pr(\sum_{i\in\overline{N}} s^{ir} \ge \sum_{i\in\overline{N}} c^{ir}) \le \frac{1}{|R|}\epsilon^{\frac{1}{M^k}}$ for all $r \in R$. Define $\epsilon^r = \frac{1}{|R|}\epsilon^{\frac{1}{M^k}}$. To achieve this objective, we use concentration inequalities [14]. Define $c^{ir} = \mu^{ir} + \beta^{ir}$, where β^{ir} is a variable to be determined for each $i \in \overline{N}$. Our objective is to ensure

$$\Pr\Big(\sum_{i\in\overline{N}}(s^{ir} - \mu^{ir}) \ge \sum_{i\in\overline{N}} \beta^{ir}\Big) \le \epsilon^r.$$

The one-sided Chebyshev's inequality [14] states that

$$\begin{aligned}
&\Pr\Big(\sum_{i\in\overline{N}}(s^{ir} - \mu^{ir}) \ge \sum_{i\in\overline{N}} \beta^{ir}\Big) \\
&\le \frac{\sum_{i\in\overline{N}}(\sigma^{ir})^2}{\sum_{i\in\overline{N}}(\sigma^{ir})^2 + \big(\sum_{i\in\overline{N}} \beta^{ir}\big)^2}.
\end{aligned}$$

Thus it suffices to ensure the following inequality holds:

$$\frac{\sum_{i\in\overline{N}}(\sigma^{ir})^2}{\sum_{i\in\overline{N}}(\sigma^{ir})^2 + \big(\sum_{i\in\overline{N}} \beta^{ir}\big)^2} \le \epsilon^r.$$

Rearranging the equation and using the fact that $\sum_{i\in\overline{N}}(\sigma^{ir})^2 \le (\sum_{i\in\overline{N}} \sigma^{ir})^2$, we obtain

$$\sum_{i\in\overline{N}} \beta^{ir} \ge \sqrt{\frac{1-\epsilon^r}{\epsilon^r}} \sum_{i\in\overline{N}} \sigma^{ir}.$$

Thus Equation (15) holds by setting $\beta^{ir} = \sqrt{\frac{1-\epsilon^r}{\epsilon^r}}\,\sigma^{ir}$ for each task $i \in \overline{N}$. The result follows. □

Theorem 1 provides a bound on selecting container size for CBS. For instance, if we want to achieve $\epsilon = 0.01$ for $M^k = 100, |R| = 2$, then Theorem 1 states that we can set container size of task type k to $\mu^{kr} + 1.1\sigma kr$, which is typically much smaller than the maximum possible size for task type k. In practice, we can use the sample mean and standard deviation to approximate the values of μ^{kr} and σ^{kr} for each $1 \le k \le K$. This is reasonable because there is usually a large number of samples per task class. Assume each sample is drawn independently, the sample mean and sample standard deviation will be close to the true mean and the true standard deviation. Finally, if a task still cannot be scheduled immediately, Harmony will keep the task in the front of the scheduling queue until it finds a machine with sufficient resources to schedule the task. This simple policy can achieves low scheduling delay as we shall demonstrate in Section 9.2.

7.3.2 Solution Algorithm

In order to leverage containers for task scheduling, we present an alternative way to round the fractional solution of $DCP - RELAX$. The idea is to leverage the following property of the first-fit (FF) algorithm:

Lemma 1. *Given a fractional solution of $DCP - RELAX$ with z_t^{m*} type m machines and $x_t^{mn^*}$ type n containers, the first-fit algorithm can place at least $\lfloor \frac{x_t^{mk}}{2|R|} \rfloor$ of each type of container n in $z_t^{m*} + 1$ machines.*

Proof. We rely on the property that the FF algorithm produces a solution in which at most one machine i is less than "half-full" (i.e., utilization $u_t^{ir} \le \frac{1}{2} \forall r \in R$). To see this, suppose this statement is false, i.e., there are two non-empty $i, j \in N_t^m$ that are less than "half-full" and i is filled before j. In this case, when FF tries to pack a container that belongs to j in the solution, it would pack it in i instead. As a result, machine j should hold no containers, which contradicts our assumption. Therefore, given a machine i with utilization u_t^{ir} for resource type $r \in R$, define the *effective utilization* of i as $\frac{1}{|R|}\sum_{r\in R} u_t^{ir}$. Based on this "half-full" property, FF ensures every machine has effective utilization at least $\frac{1}{2|R|}$ except the last non-empty machine.

Given $x_t^{mk^*}$ type n containers for each $n \in N$ that can be scheduled on z_t^{m*} type m machines, the sum of the total effective utilization must be less than z_t^{m*} as it is the maximum possible utilization for z_t^{m*} machines. Now, suppose we scale down the number of type n containers to $\lfloor \frac{x_t^{mk*}}{2|R|} \rfloor$ for each $n \in N$, the total utilization of machines is thus at most $\frac{z_t^{m*}}{2|R|}$. Suppose there are still containers waiting to be scheduled after using $z_t^{m*} + 1$ machines. As FF ensures every machine has effective utilization at least $\frac{1}{2|R|}$ except the last one, the total utilization of the $z_t^{m*} + 1$ machines is at least $\frac{z_t^{m^*}}{2|R|}$, which contradicts that the total utilization is at most $\frac{z_t^{m^*}}{2|R|}$. □

Lemma 1 essentially states that, given a fractional solution of $DCP - RELAX$ that uses z_t^{m*} type m machines and $x_t^{mk^*}$ type n containers, FF can ensure that at least $\lfloor \frac{x_t^{mk}}{2|R|} \rfloor$ containers can be placed in $z_t^{m*} + 1$ machines. Using this result, we devise our CBS algorithm (Algorithm 1) as follows: When the control interval t starts, the controller uses the predicted values $N_{t+i|t}^k \forall k \in K, 1 \le i \le W$[2] to solve $DCP - RELAX$, which gives $z_{t|t}^{m*}$, the number of active type m machines to be made available at time t. Then the controller computes an integer solution by first reducing the number of type n containers to at most $\lfloor \frac{x_t^{mk}}{2|R|} \rfloor$ and then adding containers using FF to ensure the number of type k containers is at least $\lfloor \frac{x_t^{mk*}}{2|R|} \rfloor$ for all $1 \le k \le K$. Container reassignment (i.e., migration) is then performed to ensure there are at most $z_{t|t}^{m*} + 1$ active machines. In our formulation, container reassignment cost is modeled as part of the machine switching cost, as it is only used to allow

2. We use $(t + i|t)$ to denote future value for time $t + i$ either predicted or computed at time t.

machines to be turned off. The average switching cost can be obtained through experiments. Once the container reassignment is completed and there is still room for more containers, the controller is free to schedule additional containers as long as the total number of type k containers is at most x_t^{mk}. Finally, the controller will realize the new configuration by actually turning off unused machines and making container allocations.

Theorem 2. *The integer solution produced by Algorithm 1 ensures* $U_t^{pref} - E_t - C_t^{sw} \geq (\frac{1}{2|R|} - \epsilon)U_t^{pref*} - (1+\epsilon)(E_t^* - C_t^{sw*})$ *when* z_t^m *is sufficiently large for all* $m \in M, t \in T$.

Proof. Since the number of machines used is determined by $DCP - RELAX$, it is clear that the $C_t^{sw} = C_t^{sw*}$ and $E_t^{idle} = E_t^{idle*}$. As the number of type n containers scheduled on type m machines is upper-bounded by x_t^{mk*}, we have $E_t^{util} \leq E_t^{util*}$. Finally, by Lemma 1, it is easy to show that $\lfloor \frac{x_t^{mk*}}{2|R|} \cdot \frac{z_t^{m*}-1}{z_t^{m*}} \rfloor$ containers of each type $n \in N$ can be packed in z_t^{m*} machines. As $f(\cdot)$ is a convex function, it must hold that $U_t^{pref} \geq (\max_m\{\frac{z_t^{m*}-1}{z_t^{m*}}\} - \epsilon') \cdot \frac{1}{2|R|} U_t^{pref*}$, where $\epsilon' = \max_m\{\frac{x_t^{mk*}}{2|R|} \cdot \frac{z_t^m - 1}{z_t^m} - \lfloor \frac{x_t^{mk*}}{2|R|} \cdot \frac{z_t^m - 1}{z_t^m} \rfloor\}$ is the rounding error. The theorem is proven by defining $\epsilon = \max_m\{\frac{1}{z_t^m}\} + \epsilon'$ and summing the above equations. □

Theorem 2 provides a bound on the worst case performance of CBS. In the experiments, we have observed Algorithm 1 typically performs much better than the worst case bound. Furthermore, realizing the bin-packing solutions often cannot fully utilize the machine capacities, similar to CBP, we can use a provisioning factor $\omega^k \in \mathbb{R}^+$ to account for the bin-packing inefficiencies. To account for ω^k, it suffices to replace constraint (12) by constraint (14). and run Algorithm 1 to find a suitable container placement. However, using ω^k does not lead to a better performance guarantee. To see this, consider an example where N_t^m of type m machines that are selected by $DCP - RELAX$ are active. All other machines are inactive and have $E^{idle} \approx \infty$. In this case, no matter how we adjust the value of ω^k, the number of containers scheduled by the algorithm will not improve.

Algorithm 1 Controller Algorithm for CBS

1: Provide initial state z_0^m, x_0^{mk}, $t \leftarrow 0$
2: **loop**
3: At beginning of control period t:
4: Predict $N_{t+i|t}^k$, $p_{t+i|t}$ for horizons $t = 1, \ldots, W$ using a demand prediction model
5: Solve $DCP - RELAX$ to obtain $\delta_{t+i|t}^m$,$\sigma_{t+i|t}^{mk}$ for $i = 0, \cdots, W-1$
6: Sort new containers based on their utilities
7: **for** $m \in M$ **do**
8: Select $z_{t|t}^m$ machines of type m as active machines
9: **end for**
10: Compute a re-packing configuration for all selected active machines
11: Turn on selected machines, perform re-parking using FF, turn off other machines
12: $t \leftarrow t+1$
13: **end loop**

8 Discussion

In this section we discuss considerations related to the deployment of Harmony in practice.

8.1 Task Classification and Prediction

It should be mentioned that many public cloud providers today (e.g., Amazon EC2 [1]) already offer VMs in distinct types. In such a case, our DCP algorithms can be applied directly to these public clouds. However, we argue that predefined VM sizes may not match the actual need of each customer in all cases. This is reflected by the fact that workload heterogeneity is prevalent in private clouds such as Google's compute clusters, where customers are given the flexibility to choose desired VM size. In these cases, our approach is more flexible and can provide highly efficient solution for DCP for arbitrary workload compositions.

Another important issue concerns the accuracy of the demand prediction. Even though previous work [29] suggests that ARIMA can forecast future demand with high accuracy when the trend of resource demand is stable. It is still insufficient when an unexpected demand spike occurs. In this case, we can minimize the risk of under-provisioning using the over-provisioning factor ω^k. The exact value of over-provisioning factor can be set based on experience. In Section 9.2 we shall evaluate the impact of the over-provisioning factor on the performance of CBS and CBP using the Google Traces.

8.2 Comparing CBS and CBP

Although CBS provides a theoretically-sound solution for DCP, it requires the scheduler to adopt a container-based scheduling algorithm, which is not always available in practice. As many production cloud systems (e.g., Google's compute cluster) have also developed sophisticated scheduling algorithms, implementing CBS requires major change to the design of the scheduler. On the other hand, CBP does not suffer from this limitation. However, due to lack of control of the scheduler, we have found CBP often produce worse task scheduling delay compared to CBS in our experiments. Nevertheless, in the next section we will present our evaluation of both methods and quantitatively analyze the benefits and limitations of both designs.

9 Simulation Studies

We have implemented both CBS and CBP in Matlab and studied their performance. In our implementation, the simulator receives job requests from the Google workload traces. Each job request includes information about tasks resource demand, priority, and arrival time. It then performs labeling and job scheduling according to the algorithms described in Section 7. The DCP procedure is performed once every 5 minutes, as suggested by the Google workload traces. We have chosen the time horizon to be $T = 1$, as one step prediction already provides decent performance in our experiments.

In our experiments, we simulate a heterogeneous cluster composed of a mixture of servers from multiple manufacturers and models. Table 2 provides the characteristics of the simulated servers. We normalized the CPU core count

TABLE 2
Machine Configurations

Model	Num. of Processors	Num. of Cores	Memory Memory	Num. of Machines
Dell PowerEdge R210	1	4	4 GB	7000
Dell PowerEdge R515	2	6	32 GB	1500
HP DL385 G7	2	12	16 GB	1000
HP DL585 G7	4	12	64 GB	500

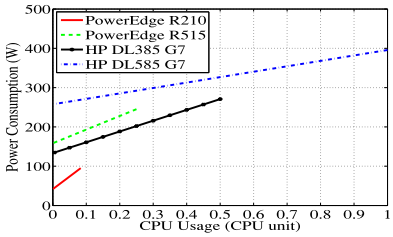


Fig. 9. Machine energy consumption rate.

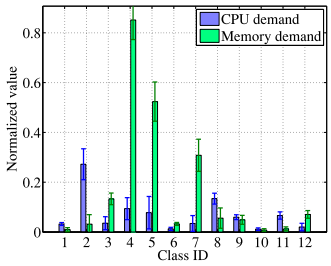


Fig. 10. Class size (Gratis).

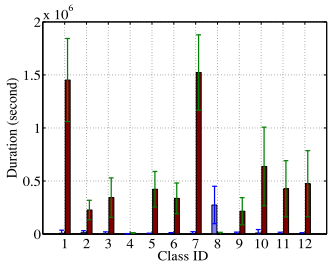


Fig. 11. Task duration (Gratis).

and memory capacity to the largest machine size. Hence, HP DLG585 G7 has a capacity 1 CPU unit and 1 memory unit, which corresponds to 48 cores and 64 GB, respectively. The energy consumption of the different machines is modeled according to Equation (6). The parameters $E^{idle,m}$ and α^{mr} for each type of servers were estimated using energy measurements available in [2]. Furthermore, we use the average electricity price in the state of California, which is 12.25 cents per Kilowatthour [6]. The reconfiguration cost per machine power cycle is set to 0.5 cents [12]. Energy costs reported hereafter include cooling costs, which are considered to be proportional to the cost of energy consumed by the servers (the proportionality factor is set to 0.8) [21].

Fig. 9 shows the energy consumption as function of CPU usage. Indeed, this figure demonstrates the importance of considering the machine heterogeneity when scheduling tasks in order to reduce energy consumption. For instance, a container requiring 0.2 CPU units should be placed in a HP DL385 G7 since the PowerEdge R210 does not have

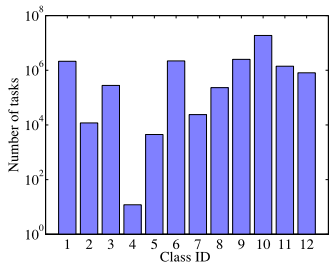


Fig. 12. Number of tasks (Gratis).

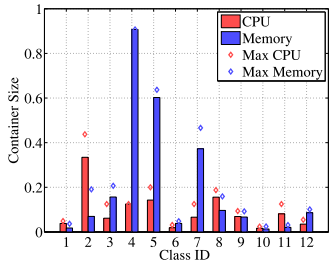


Fig. 13. Container size versus maximum size (Gratis).

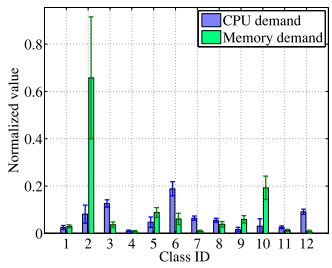


Fig. 14. Class size (other).

enough CPU capacity, whereas the other types of servers are able to host it but will consume much more energy. Selecting the "right" machines to switch on becomes particularly challenging when millions of heterogeneous tasks have to be scheduled in the cluster.

9.1 Results of Task Classification

We performed task classification as described in Section 5.1. For each priority group, we varied the value of k and evaluated the quality of the resulting clusters produced by the K-means algorithm. The best value of k for each priority group is selected as the one for which no significant benefit can be achieved by increasing the value of k. The results after the first step of our characterization for each priority group are shown in Figs. 10, 14, and 18, respectively. These diagrams show the clustering algorithm captures the differences in task sizes and identifies cpu-intensive tasks and memory-intensive tasks. Furthermore, the standard deviation is much less than the mean value for both CPU and memory, which confirms the accuracy of the characterization. The number of tasks in each task class is shown in Figs. 12, 16 and 20, respectively. It is clear that the number of tasks within each cluster can vary significantly. Most of the classes have between 10^4 and 10^6 tasks except cluster 4 for Gratis priority group, which has only 100 tasks. Lastly,

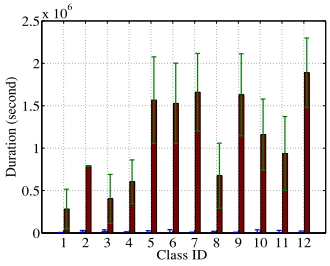


Fig. 15. Task duration (other).

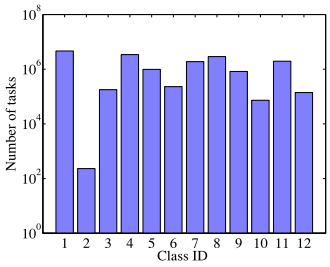


Fig. 16. Number of tasks (other).

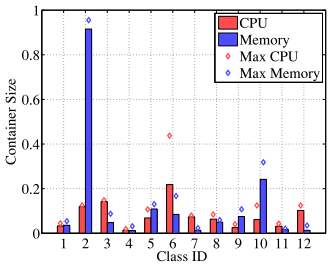


Fig. 17. Container size versus maximum size (other).

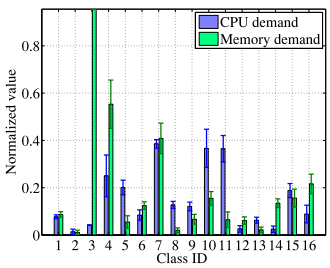


Fig. 18. Class size (production).

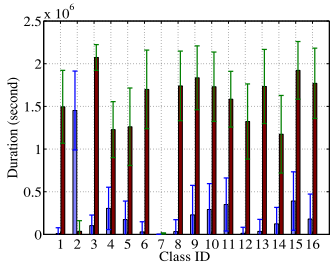


Fig. 19. Task duration (production).

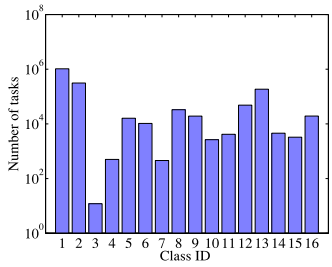


Fig. 20. Number of tasks (production).

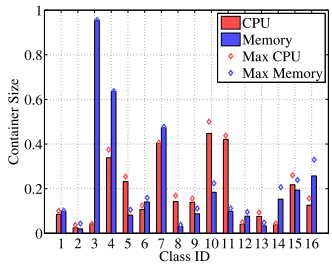


Fig. 21. Container size versus maximum size (production).

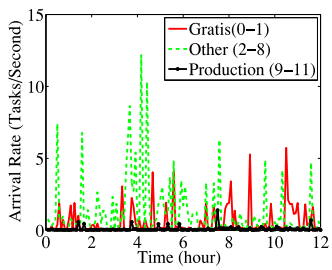


Fig. 22. Aggregated task arrival rates.

we run the k-means algorithm with $k = 2$ to categorize tasks of each task class as either short or long. The results are shown in Figs. 11, 15 and 19, respectively. These diagrams confirm that long tasks typically run several orders of magnitude longer than short tasks. Finally, we computed the container size as described in Section 7.3.1, with $\epsilon = 0.001$, $|R| = 2$ and $M^k = 100$. The results are shown in Figs. 13, 17 and 21, respectively. Clearly, the size of containers is typically smaller than the maximum task size within the cluster.

9.2 Controller Performance

We have evaluated the performance of CBS and CBP algorithms using Google workload traces. In our experiments, the sum of arrival rate of tasks belonging to each priority group is shown in Fig. 22. Fig. 23 shows the sum of the total containers belonging to each priority group computed by Harmony.

For comparison purpose, we also implemented a baseline (heterogeneity-oblivious) algorithm that tries to find a balance between energy savings and scheduling by maintaining an 80 percent utilization of the bottleneck resource. Essentially, given the total resource demand, the algorithm provisions machines in a "greedy" fashion by turning them on in decreasing order of energy efficiency (e.g., always turning on HP-DL585-G7 machines first). We picked the value of 80 percent because we found that a utilization higher than 80 percent can cause a significant increase in task scheduling delay. As the Google workload contains many long running tasks that were scheduled before the start of the traces, in our

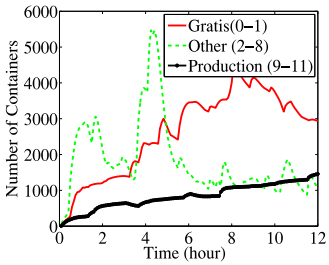


Fig. 23. Number of required containers.

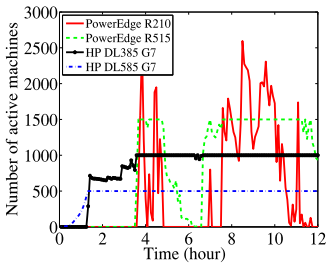


Fig. 24. Number of machines used by the baseline.

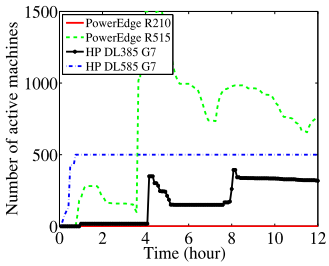


Fig. 25. Number of machines used by CBS/CBP.

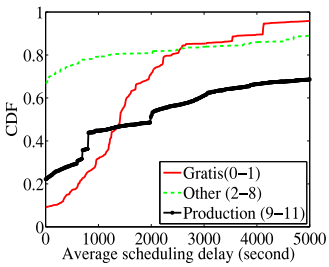


Fig. 26. CDF of scheduling delay for baseline.

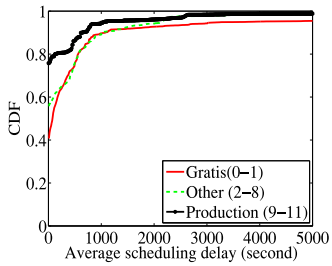


Fig. 27. CDF of scheduling delay for CBP.

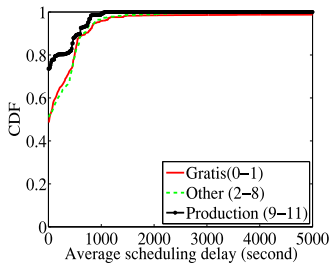


Fig. 28. CDF of scheduling delay for CBS.

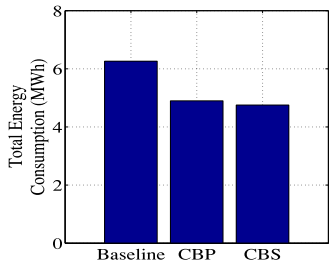


Fig. 29. Comparison of energy consumption.

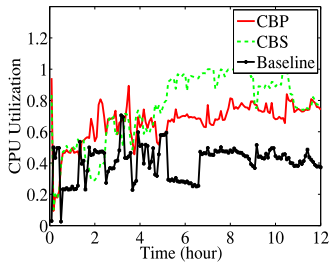


Fig. 30. CPU utilization in the data center.

simulation, we mainly focus on simulating the arrival of new tasks.

In our first experiment, we use an over-provisioning factor of 1.2 to demonstrate the behavior of our algorithms. The number of active servers provisioned by the baseline algorithm, CBS and CBP are shown in Figs. 24, 25 respectively. Note that both CBS and CBP provision the same number of machines as indicated by the MPC algorithm. It can be seen that the number of machines provisioned by CBS and CBP is much less than the number of machines selected by the baseline algorithm. Furthermore, It can be seen that they are able to make intelligent decisions regarding what type of machines to turn on and off. Fig. 29 shows the total energy consumption of all three approaches. It can be seen that CBS incurs the lowest energy costs, corresponding to a 28 percent reduction in energy cost compared to the baseline algorithm.

The CDF of task scheduling delays are shown in Figs. 26, 27 and 28, respectively. It can be seen that CBS and CBP can substantially reduce the scheduling delay compared to the baseline algorithm. The CPU and memory utilizations of the baseline, CBS and CBP are compared in Figs. 30 and 31, respectively. It can be seen from the diagrams that the baseline achieves low utilization for both CPU and memory. This is because the baseline only ensures the total provisioned capacity is $\frac{1}{80\ \text{percent}}$ times the required capacity, and does not consider the types of machines provisioned. As soon as the most energy efficient (i.e., HP DL585 G7) machines are all turned on, it began to make wrong

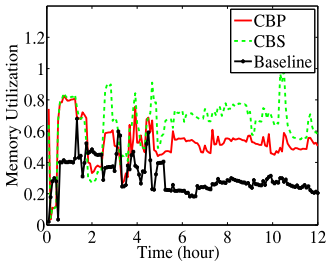


Fig. 31. Memory utilization in the data center.

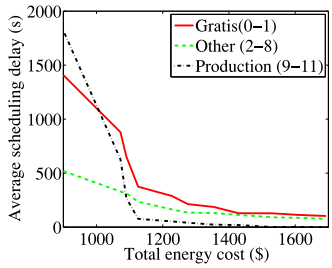


Fig. 33. Energy cost versus scheduling delay for CBS.

decisions regarding the type of machines to be turned on. As a result, many tasks cannot be scheduled in the newly provisioned machines, resulting in low utilization and high scheduling delay. In contrast, both CBS and CBP can significantly outperform the baseline algorithm in terms of both resource utilization and scheduling delay. Furthermore, CBS generally outperform CBP in our experiments. This is because CBS uses dedicated containers for scheduling, thus ensuring that large tasks can be scheduled quickly. In contrast, CBP does not provide guaranteed resources for large tasks, making them more difficult to schedule.

To better understand the difference between CBS and CBP, we varied the values of the over-provisioning factor ω^k between 0.6-2.2 to produce different tradeoffs between operational costs (energy and reconfiguration cost) and the average scheduling delay. The results are shown in Figs. 32 and 33, respectively. We found when the cluster is under-provisioned ($\omega^k \leq 1.1$), scheduling delays are high for both schemes. In this case, CBS performs poorly because it can only schedule tasks in dedicated containers. If all dedicated containers reserved for a task class run out, a task that belongs to the class has to wait even if there are idle resources in the cluster. In contrast, CBP does not have this limitation, because a task can be scheduled whenever there is sufficient resources available in the cluster. However, this is no longer true when $\omega^k \geq 1.1$. In this case, CBS outperforms CBP because dedicated containers can guarantee every type of tasks can be scheduled without much delay, whereas the bin-packing algorithm used by CBP can have difficulties scheduling large tasks, especially production tasks. These observations suggest that CBS can slightly outperform CBP in terms of solution quality. However, as CBS adopts a strict (i.e., slot-based) scheduling policy, it is less practical for real-world deployment. Thus, the cloud provider must carefully analyze these tradeoffs in order to decide which scheme should be used in a given scenario.

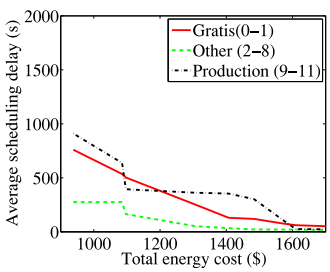


Fig. 32. Energy cost versus scheduling delay for CBP.

10 Conclusion

Dynamic capacity provisioning has become a promising solution for reducing energy consumption in data centers in recent years. However, existing work on this topic has not addressed a key challenge, which is the heterogeneity of workloads and physical machines. In this paper, we first provide a characterization of both workload and machine heterogeneity found in one of Google's production compute clusters. Then we present Harmony, a heterogeneity-aware framework that dynamically adjusts the number of machines to strike a balance between energy savings and scheduling delay, while considering the reconfiguration cost. Through experiments using Google workload traces, we found Harmony yields large energy savings while significantly improving task scheduling delay.

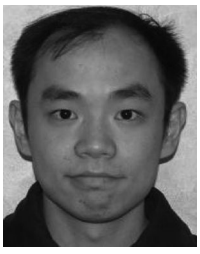
Acknowledgments

This work was supported in part by the Natural Science and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network, and in part by a Google Faculty Research Award.

References

[1] Amazon Elastic Computing Cloud, http://aws.amazon.com/ec2/, 2013.
[2] Energy Star Computer Server Qualified Product List, energystar.gov/ia/products/prod_lists/enterprise_servers_prod_list.xls, 2014.
[3] Eucalyptus community, http://open.eucalyptus.com/, 2014.
[4] Googleclusterdata - traces of google workloads, http://code.google.com/p/googleclusterdata/, 2014.
[5] Technology research - Gartner Inc, www.gartner.com, 2014.
[6] U.S. Energy Information Adminstration, http://www.eia.gov/, 2014.
[7] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective Straggler Mitigation: Attack of the Clones," *Proc. 10th USENIX Conf. Networked Systems Design and Implementation (NSDI)*, 2013.
[8] R. Boutaba, L. Cheng, and Q. Zhang, "On Cloud Computational Models and the Heterogeneity Challenge," *J. Internet Services and Applications*, vol. 3, pp. 77-86, 2012.
[9] G.E.P. Box, G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis, Forecasting, and Control*. Third ed., Prentice-Hall, 1994.
[10] S. Boyd et al., *Convex Optimization*. Cambridge Univ. Press, 2004.
[11] C. Chekuri and S. Khanna, "On Multi-Dimensional Packing Problems," *Proc. Symp. Discrete Algorithms*, 1999.
[12] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing Server Energy and Operational Costs in Hosting Centers," *ACM SIGMETRICS Performance Evaluation Rev.*, vol. 33, pp. 303-314, 2005.
[13] Y. Chen et al., "Analysis and Lessons from a Publicly Available Google Cluster Trace," Technical Report UCB/EECS-2010-95, 2010.
[14] J. Diaz et al., "A Guide to Concentration Bounds," *Handbook on Randomized Computing*. Springer, 2001.

[15] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types," *Proc. Eighth USENIX Conf. Networked Systems Design and Implementation (USENIX NSDI)*, 2011.
[16] D. Gross and C. Harris, *Fundamentals of Queueing Theory*, pp. 244-247, John Wiley & Sons, 1998.
[17] G. Jung, M.A. Hiltunen, K.R. Joshi, R.D. Schlichting, and C. Pu, "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures," *Proc. IEEE 30th Int'l Conf. Distributed Computing Systems (ICDCS)*, 2010.
[18] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubrahmanian, K. Talwar, L. Uyeda, and U. Wieder, "Validating Heuristics for Virtual Machines Consolidation," Microsoft Research, MSR-TR-2011-9, 2011.
[19] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic Right-Sizing for Power-Proportional Data Centers," *Proc. IEEE INFOCOM*, 2011.
[20] A.K. Mishra, J.L. Hellerstein, W. Cirne, and C.R. Das, "Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters," *ACM SIGMETRICS Performance Evaluation Rev.*, vol. 37, pp. 34-41, Mar. 2010.
[21] C.D. Patel and A.J. Shah1, "Cost Model for Planning, Development and Operation of a Data Center," Technical Report HPL-2005-107(R.1), HP Laboratories Palo Alto, 2005.
[22] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the Electric Bill for Internet-Scale Systems," *Proc. ACM SIGCOMM*, 2009.
[23] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozuch, "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis," *Proc. ACM Symp. Cloud Computing*, 2012.
[24] S. Ren et al., "Provably-Efficient Job Scheduling for Energy and Fairness in Geographically Distributed Data Centers," *Proc. IEEE 32nd Int'l Conf. Distributed Computing Systems (ICDCS)*, 2012.
[25] B. Sharma, V. Chudnovsky, J.L. Hellerstein, R. Rifaat, and C.R. Das, "Modeling and Synthesizing Task Placement Constraints in Google Compute Clusters," *Proc. Second ACM Symp. Cloud Computing (SOCC)*, 2011.
[26] A. Verma et al., "Pmapper: Power and Migration Cost Aware Application Placement in Virtualized Systems," *Proc. Ninth ACM/IFIP/USENIX Int'l Conf. Middleware (Middleware)*, 2008.
[27] Q. Zhang, J. Hellerstein, and R. Boutaba, "Characterizing Task usage Shapes Google's Compute Clusters," *Proc. LADIS Workshop Held in Conjunction with VLDB*, 2011.
[28] Q. Zhang, M.F. Zhani, R. Boutaba, and J.L. Hellerstein, "HARMONY: Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud," *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS)*, 2013.
[29] Q. Zhang, M.F. Zhani, Q. Zhu, S. Zhang, R. Boutaba, and J.L. Hellerstein, "Dynamic Energy-Aware Capacity Provisioning for Cloud Computing Environments," *Proc. ACM Int'l Conf. Autonomic Computing (ICAC)*, 2012.



Qi Zhang received the BASc, MSc, and PhD degrees from the University of Ottawa, Canada, Queen's University, Canada, and the University of Waterloo, Canada, respectively. His current research focuses on resource management for cloud computing systems. He is also interested in related areas including big-data analytics, software-defined networking, network virtualization and management. He is a student member of the IEEE.

Mohamed Faten Zhani received the engineering and MS degrees from the National School of Computer Science, Tunisia in 2003 and 2005, respectively. He received the PhD degree in computer science from the University of Quebec in Montreal, Canada in 2011. Since then, he has been a postdoctoral research fellow at the University of Waterloo. His research interests include cloud computing, virtualization, Big data and software defined networking. He is a member of the IEEE.

Raouf Boutaba received the MSc and PhD degrees in computer science from the University Pierre and Marie Curie, Paris, France, in 1990 and 1994, respectively. He is currently a professor of computer science with the University of Waterloo, Waterloo, ON, Canada. His research interests include control and management of networks and distributed systems. He is a fellow of the IEEE and the Engineering Institute of Canada.

Joseph L. Hellerstein received the PhD degree in computer science from UCLA. He was a principal architect at Microsoft Corp. in Redmond (2006-2008), and a researcher and a senior manager at the IBM Thomas J. Watson Research Center in Hawthorne (1984-2006). He manages the Computational Discovery Department at Google Inc. in Seattle, WA. He has published more than 100 peer-reviewed papers and two books, and has taught at Columbia University and the University of Washington. He is a fellow of the IEEE.