HARMONY: Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud

Qi Zhang, Mohamed Faten Zhani, Raouf Boutaba
University of Waterloo
{q8zhang, mfzhani, rboutaba}@uwaterloo.ca

Joseph L. Hellerstein
Google Inc.
jlh@google.com

***Abstract*—Data centers today consume tremendous amount 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 for dynamic capacity provisioning have not fully considered the heterogeneity of both workload and machine hardware found in production environments. In particular, production data centers often comprise several generations of machines with different capacities, capabilities and energy consumption characteristics. Meanwhile, the workloads running in these data centers typically consist of a wide variety of applications with different priorities, performance objectives and resource requirements. Failure to consider heterogenous characteristics 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, in this paper we present HARMONY, a Heterogeneity-Aware Resource Management System for dynamic capacity provisioning in cloud computing environments. Specifically, we first use the K-means clustering algorithm to divide the workload into distinct task classes with similar characteristics in terms of resource and performance requirements. Then we present a novel technique for dynamically adjusting the number of machines of each type to minimize total energy consumption and performance penalty in terms of scheduling delay. Through simulations using real traces from Google's compute clusters, we found that our approach can improve data center energy efficiency by up to 28% compared to heterogeneity-oblivious solutions.**

I. 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 large number of machines, they also incur tremendous energy costs in terms of power distribution and cooling. In particular, 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% reduction in energy cost can translate to over a million dollars in cost savings [16]. On the other hand, governmental agencies continue to implement standards 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 [18], [22]. 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* [21], [19], [15], [17], which is the time a request has to wait 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 trade-off 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.

However, 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 [17]. 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 [19]. They have heterogeneous processing capacities and capabilities, different hardware features, processor architecture, processor speed, memory and disk size. Consequently, they also have different energy consumption rates at run-time.
- **Workload Heterogeneity**. Production data centers typically receive vast number of heterogeneous resource requests with diverse resource demand, durations, priorities and performance objectives [21], [19], [15]. In particular, it has been reported that the difference in resource demand and duration can span several orders of magnitude [21], [19], [6].

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 surge of workload requests, a heterogeneous-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 is known to be difficult. In particular, designing such a scheme requires accurate characterization of both workload and machine heterogeneities. It also requires a heterogeneity-aware performance model that strikes a balance between energy savings and scheduling delay at run-time.

In this paper, we present HARMONY, a **H**eterogeneity-**A**ware **R**esource **MON**itoring and management s**Y**stem that addresses the aforementioned challenges for DCP. 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 distinct 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 run-time electricity prices. We then devise an online control algorithm based on the *Model Predictive Control* framework that dynamically adjusts the number of servers in order to minimize total energy cost and task scheduling delay, while taking into account the switching costs of machines.

The remainder of the paper is organized as follows. Section II surveys related work in the literature. Section III provides an analysis of a publicly available workload traces from Google to motivate our approach. Section IV provides an overview of HARMONY. In Section V, VI and VII we describe the design of HARMONY in details. Section VIII discusses the deployment of HARMONY in practice. Finally, we evaluate our proposed system using Google workload traces in Section IX, and draw our conclusions in Section X.

II. Related work

In this section we provide a survey of existing studies on (1) understanding workload and machine characteristics in production clouds, and (2) dynamic capacity provisioning for balancing the trade-off between energy savings and application performance objectives.

A. Machine and workload characterization

Both capacity planning and task scheduling require a deep analysis of the workload characteristics in terms of arrival rate, requirements, and duration [15]. As a result, characterizing workload in production clouds has received much attention in recent years. 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 [15]. Following the same line of research, Chen et. al. provided a characterization of Google cluster workload at job-level applying the k-means algorithm [10]. Sharma et. al. [19] and Zhang et. al. [21] studied the problem of finding accurate workload characterizations through benchmark generation and validation. Recently, Reiss et. al. [17] provided a comprehensive analysis of the heterogeneity and dynamicity found in Google cluster traces. They have shown 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.

B. Energy-aware capacity provisioning

There is a large body of literature on energy-aware dynamic capacity provisioning in data centers. For example, pMapper [20] 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. Similarly, Mistral [14] is a framework that dynamically adjusts VM placement to find a trade-off 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. [18] studied the problem of scheduling heterogenous batch workload across geographically distributed data centers. Different from out work, they assume workload has already been divided into distinct types. Furthermore, they do not consider schedulability issues, since for general cloud workloads, not every task can be divided arbitrarily and scheduled on any machine. 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.

III. 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 [3][1] consisting of approximately $12,000$ machines. The workload traces contain scheduling events as well as 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. The values of the demand for each resource type were normalized between 0 and 1. Even though the dataset 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)

[1]It should be mentioned that the same dataset has been analyzed by Reiss et. al.[17]. However, our analysis extends, and largely complements the results in [17].

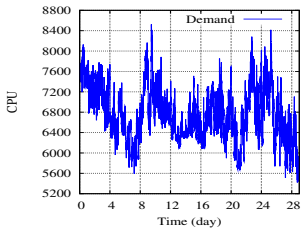


Fig. 1: Total CPU demand

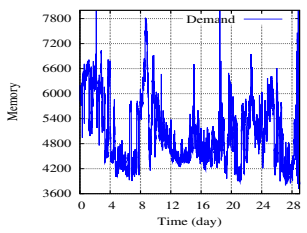


Fig. 2: Total Memory demand

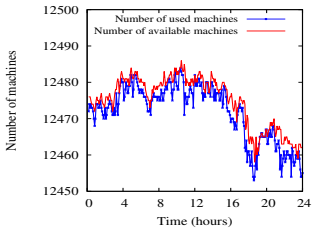


Fig. 3: Number of machines

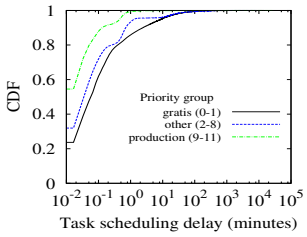


Fig. 4: CDF Scheduling delay

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*$(9-11)$ [17]. In this paper, we primarily analyze task characteristics at the priority group-level, because priority groups already provide a coarse-grained classification of tasks. Furthermore, they also have strong correlation with task scheduling classes [3], [17]. Nevertheless, our technical approach can be extended to handle any combination of task priority groups and task scheduling classes. 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.

A. 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 Figure 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. Figure 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. Figure 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. These observations suggest that a large number of machines can be turned off to save energy.

B. 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. Figure 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% of the tasks in *production* and *other* priority groups respectively are scheduled immediately. However, 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 there is a constraint that is difficult to satisfy. These results suggests that more efficient provisioning and scheduling methods are needed to reduce the scheduling delay for these difficult-to-schedule tasks.

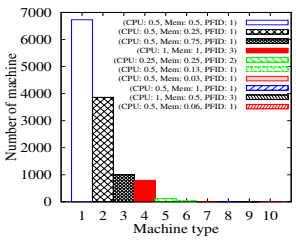


Fig. 5: Machine heterogeneity in compute cluster

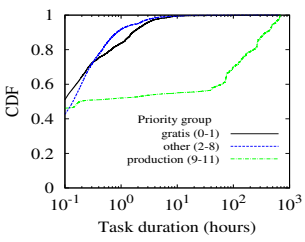


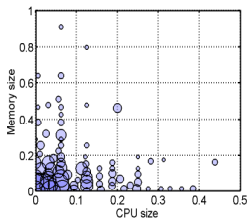
Fig. 6: CDF of task duration for different priority groups

C. Understanding Machine Heterogeneity

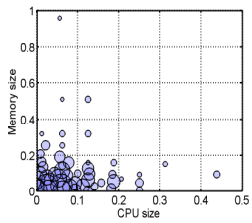
The dataset also provides information about the types of machines running in the cluster. A machine is characterized by its capacity in terms of CPU, memory and disk size as well as a platform ID, which identifies the micro-architecture (e.g., vendor name and chipset version) and memory technology (e.g., DDR or DDR2) of the machine. Similar to tasks, machine capacities are normalized such that the largest machine has a capacity equal to 1. Figure 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% of the machines belong to machine types 1 and 2, respectively. On the other hand, machines types 3 and 4 have around 1000 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 clear that such a heterogeneity will translate into different energy consumption models.

D. Understanding Task Heterogeneity

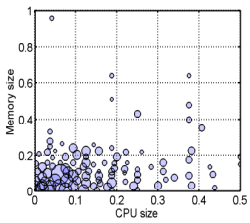
In order to analyze the workload heterogeneity, we plotted tasks requirements and their durations for the three priority groups. Figure 7 shows the CPU and memory size of tasks belonging to each priority group. The coordinates of each



(a) Gratis (0-1)



(b) Other



(c) Production

Fig. 7: Task Size Analysis

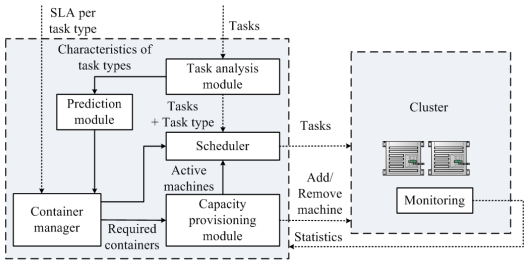


Fig. 8: System architecture

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% 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 requirement and memory requirements. Another key observation is that the difference in task size can span several orders of magnitude. For example, Figure 7a shows that the largest task in the gratis priority group is almost $1000\times$ bigger than the smallest task in the same group for both CPU and memory. Similar characteristics can also be found in Figure 7b and 7c. Finally, by comparing Figure 5 and Figure 7, it is easy to see that not every task (e.g., CPU size ≈ 1) can be scheduled on every type of machine (e.g., CPU capacity$= 0.5$).

Another important parameter that shows the heterogeneity of the tasks is the task duration. Figure 6 shows the CDF of task durations for tasks with different priority groups. From Figure 6, it can be seen that production tasks (9-11) have long durations that can reach 17 days, whereas 90% of the remaining tasks (i.e., gratis and other) have shorter duration that ranges between 0 and 10 hours. The same observation can be done for production-priority tasks when compared to other priority groups (Figure 6). Furthermore, it is worth noting that more than 50% of the tasks are short (less than 100 seconds). This concurs with the previous workload analysis studies [21], which showed that tasks are either short or long.

E. 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 task scheduling and capacity provisioning scheme is challenging, as it involves finding a satisfactory compromise between energy savings and scheduling delay, given 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 [6]. 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 for DCP.

IV. System Overview

As discussed previously, we aim at designing 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 run time, and (3) using the captured information to control the number of machines in the compute cluster to achieve a balance 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 for monitoring and computation. Thus, a medium-grained characterization of the workload is necessary. Motivated by this observation, we present a workload characterization of Google traces by dividing tasks into *task classes* using the K-means algorithm. However, different from previous work on this topic [15], [10] that focuses on only understanding workload characteristics, our goal is to ensure high accuracy of the characterization, while supporting efficient task classification (e.g., labeling) at run time. It should be mentioned that machines are naturally characterized (i.e., there are 10 types of machines in the cluster). Thus, in our solution we will mainly focus on task characterization.

Once we have obtained the workload characterization, we introduce a monitoring mechanism that allows HARMONY to capture the run-time workload composition in terms of arrival rate for each task class. To make resource allocation decisions, we then define a *container* as a logical allocation of resources to a task that belongs to a task class. In our approach, the task containers serve as reservations for helping the controller to make machine allocation decisions. It is also possible to directly use task containers for scheduling (to be described in Section VII). 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 Figure 8. It consists of the following components. *The task analysis*

module is responsible for monitoring the arrival of every task class 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 cope with the current workload. These parameters are evaluated based on two factors, namely, the predicted arrival rate and the required SLO for each type of tasks expressed in terms of average scheduling delay. The container manager periodically notifies the capacity provisioning module about the number of required containers of each type. *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 would be able to host the containers and, at the same time, minimizes the energy consumption. *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. It should be mentioned that currently we only focus on the cases where disk storage of a VM is either handled by a storage network or collocated with the VM itself. However, it is possible to extend HARMONY to consider distributed file systems such as the Google File System (GFS) [12]. In the following sections, we describe the design of HARMONY in details.

V. Task Characterization and Run-time Classification

The goal of task classification is to divide the workload into task classes that are accurate for DCP yet efficient for run-time task labeling. 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. For run-time task labeling, we can observe the task characteristics and assign it to the class that has the highest *similarity score*. Following the common approach, the similarity score between a task and a task class is computed as the Euclidean distance between the task and the centroid of the task class in the feature space.

Generally speaking, achieving a high accuracy in characterization can be realized by properly controlling the number of task classes. On the other hand, performing accurate run-time task labeling is more difficult, because task running time is generally unknown to the system until the task finishes. In Harmony, we address this issue by realizing 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.

To provide better support for this mechanism, we adopt a two-step approach for workload clustering. In the first step, tasks are classified based on static characteristics (e.g., priority, CPU and memory size specified in the job request) using k-means algorithm. In the second step, each task class is further divided into sub-classes based on task running time. The advantage of this approach is that it not only simplifies the relabeling process, but also reduces the error introduced by the relabeling process.

VI. Workload Prediction

As mentioned previously, the task analysis module is responsible for monitoring the arrival of tasks and determine their types based on their size and current running time. This also allows the task analysis module to predict the future arrival rate of tasks. Currently, we have implemented a time series-based predictor using the well-known ARIMA [7] model. Once the predictions have been obtained, the next step is to determine the number of machines of each type needed in next control period. We address this problem by computing the number of containers required to support the workload for each task class. In HARMONY, this is performed by the container manager. The container manager evaluats the number of containers per type of tasks such that the desired scheduling delay is achieved. Let c_i denote the number of containers for tasks type i such that the average scheduling delay is equal to $\bar{d}_i$. 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 [13]:

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

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. It is expressed as:

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

Given an average scheduling delay and using Eq. (1), it is easy to estimate c_i to ensure $d_i \leq \bar{d}_i$ and $\rho_i < 1$.

The number of containers computed using Eq. (2) provides an estimation of the number of required containers to guarantee performance objectives. In the next section, we discuss how to use this number for machine selection.

VII. Container-Based Scheduling for DCP

In this section. We describe our solution called *Container-Based Scheduling (CBS)*, which aims at finding an allocation of containers that can be used for run-time task scheduling.

A. Modeling Container Size

One of the main challenges for using containers for scheduling is to select appropriate container size. On one hand, setting the container size equal to the maximum value in the class ensures every task can be scheduled without violating machine capacity constraints. However, it also causes resource wastage

TABLE I: Table of Notations

Symbol	Meaning
d_i	Average scheduling delay for task class i
R	Resource types
s_i^r	Size of task i for resource type r
c_i^r	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
$\alpha^{\bar{m}r}$	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 variable indicating machine i is active at time t
u_t^i	Change in machine i's state at time t
a_t^{in}	Num. of type n containers assigned to machine i at time t
γ_t^{in}	Change in num. of type n containers on machine i at time t
z_t^m	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^{mn}	Num. of type n containers assigned to machine m at time t
σ_t^{mn}	Change in num. of type n containers in type m machines at t

due to overestimation of true task resource demand in the average case. In contrast, setting the container size equal to the average task size will lead to under-estimation of task resource assumption, resulting in machine capacity violations.

To deal with this issue, we rely on the statistical multiplexing of task resource demand to ensure the probability of machine capacity violation to be low. Specifically, it is known that k-means algorithm tries to model the input as a mixture of Gaussian distributions with identity covariance matrix [4]. Thus, it is reasonable to assume tasks in each task class n follow an independent gaussian distribution $\mathcal{N}(\mu_n^D, \sigma_n^D)$ for each resource type $r \in R$, where $R = \{1, 2, ..., D\}$ denotes the set of resource types. The problem of selecting the optimal container size can be stated as follows: For each task class $n \in N$ and resource type $r \in R$, for any machine m with resource capacities $C^{mr} \forall r \in R$ and any group of tasks G where each task $i \in G$ has actual demand s^{ir} and container size c^{ir}, find a value c^{nr} such that if $\sum_i c^{ir} \leq C^{mr}\ \forall r \in R$, then $\Pr(\sum_i s^{ir} > C^{mr} \forall r \in R) \leq \epsilon$, where ϵ is the error bound we wish to guarantee.

To solve this problem, we first translate ϵ into resource specific error bounds ϵ^r for $r \in R$ using the formula for joint probability. Assuming all tasks in G are independent, then the total task usage is also normally distributed, namely $\sum_i s^{ir} \sim \mathcal{N}(\sum_i \mu_i^r, \sum_i (\sigma^{ir})^2)$, $\forall r \in R$. Define Z_ϵ^r as the $(1-\epsilon^r)$-percentile of the unit normal distribution, our goal is to ensure

$$\frac{C^{mr} - \sum_{i \in G} \mu_i^r}{\sqrt{\sum_{i \in G} (\sigma_i^r)^2}} \geq \quad Z_\epsilon^r \tag{3}$$

holds for all $r \in R$. In this case, we set $c^{ir} = \mu^{ir} + Z_\epsilon^r \sigma^{ir}$, which satisfies the above inequality. Then, we can reserve extra machines to handle machine capacity violation by scheduling the tasks causing the violation in the reserved machines. Finally, even though we assume task size follows Gaussian distribution, the results for other clustering methods and more general distributions (i.e., non-Gaussian) can be derived similarly using concentration bounds [11].

B. CBS Formulation

We now provide a formal model for CBS. 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, each $m \in M$ has N_t^m machines available at time t (i.e., the t'th time interval). 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 N types of containers to schedule at time t, the number of containers of type $n \in N$ is N_t^n. Let $c^{nr} \in \mathbb{R}^+$ denote the size of a type $n \in N$ container for resource type $r \in R$.

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

$$y_{t+1}^i = y_t^i + u_t^i \tag{4}$$

$$a_{t+1}^{in} = a_t^{in} + \gamma_t^{in} \tag{5}$$

The utilization of machine i for type r resource at time t is

$$u_t^{ir} = \frac{1}{C^{mr}} \sum_{n \in N} z_t^{in} c^{nr}. \tag{6}$$

As total energy usage of a physical machine can be estimated by a linear function of resource utilization [22], let p_t denote the energy price at time t, the energy cost at time t is:

$$E_t = p_t \sum_{m \in M} \sum_{i \in N_t^m} y_t^i \left(E^{idle,m} + \sum_{r \in R} \alpha^{mr} u_t^{ir} \right) \tag{7}$$

where $E^{idle,m} \in \mathbb{R}^+$ is the energy consumption of a type m machine when it is idle, and $\alpha^{nr} \in \mathbb{R}^+$ is the slope of the linear 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 $E_t = E_t^{idle} + E_t^{util}$. Next, to model the scheduling performance, since it is not possible for all containers to be scheduled when demand is high, we assume there is a utility function $f^n(\cdot)$ that models the monetary gain for scheduling containers. $f^n(\cdot)$ is assumed to be a concave function that can be derived from SLO objectives (e.g., reduction in monetary cost due to scheduling delay). The total revenue becomes:

$$U_t^{perf} = \sum_{n \in N} f^n \Big(\sum_{m \in M} \sum_{i \in N_t^m} a_t^{in} \Big) \tag{8}$$

The cost for turning machines on and off can be described by:

$$C_t^{sw} = \sum_{m \in M} \sum_{i \in N_t^m} q^m |u_t^i| \tag{9}$$

where $q^m \in \mathbb{R}^+$ denotes the switching cost of a single type m machine. Finally, we require the following constraints:

$$y_t^i \geq a_t^{in} \quad \forall m \in M, i \in N_t^m, n \in N, t \in \mathcal{T} \tag{10}$$

$$\sum_{n \in N} a_t^{ni} c^{nr} \leq C^{mr} \qquad \forall m \in M, i \in N_t^m, t \in \mathcal{T} \tag{11}$$

Constraint (10) states that a machine is active if it hosts at least one container. Constraint (11) ensures that containers scheduled on the same machine do not exceed the resource capacity of the machine. Thus, the overall objective of CBS is control the number of active machines and adjust container placement in a way that minimizes the energy consumption, while minimizing the number of machines to be switched on and off over a time horizon $\mathcal{T} = \{1, 2, ..., T\}$:

$$\max_{a_t^{in}, u_i^i, y_t^i, a_t^{in}} \quad R_T = \sum_{t=1}^{T} U_t^{pref} - E_t - C_t^{sw}$$

subjects to constraints (4), (5), (10) and (11). CBS is $\mathcal{NP}$-hard to solve as it generalizes the vector bin-packing problem [9]. Furthermore, linear programming based solutions cannot be applied to CBS due to the large number of variables involved. Given 80 task classes and over $10K$ machines, CBS contains at least $800K$ variables, making it difficult to solve in online settings. Finally, traditional bin-packing heuristics (e.g., First-Fit) do not apply directly to CBS as they do not consider machine switching and container reassignment costs.

C. Solution Algorithm

To address the above limitation, we first describe a relaxed version of CBS, where the number of machines (i.e., y_{it}) and container assignment (i.e., z_t^{in}) no longer take integer values. This relaxation yields a simpler formulation, as we only need to maintain the container counts at an aggregate level (i.e., per-machine type) rather than at per-machine level. This addresses the scalability issue associated with CBS. Specifically, denote by $z_t^m \in \mathbb{R}^+$ the number of active type m machines at time t, and $\delta_t^m \in \mathbb{R}$ the change in the number of machines at time t. Similarly, define $x_t^{mn} \in \mathbb{R}^+$ as the number of type n containers assigned to machines of type m that is capable of hosting container type n, and $\sigma_t^{mn} \in \mathbb{R}^+$ the change in x_t^{mn} at time t, we have the following equations:

$$z_{t+1}^m = z_t^m + \delta_t^m \tag{12}$$
$$x_{t+1}^{mn} = x_t^{mn} + \sigma_t^{mn} \tag{13}$$

It is straightforward to show the relaxed version of CBS can be rewritten as the following problem called $CBS - RELAX$:

$$\max_{\delta_t^m, \sigma_t^{mn}} \sum_{t=0}^{T} \sum_{m \in M} -p_t \left(z_t^m E^{idle,m} + \sum_{r \in R} \sum_{n \in N} \frac{\alpha^{mr} c^{nr}}{c^{mr}} \cdot x_t^{mn} \right) + \sum_{n \in N} f^n \Big(\sum_{m \in M} x_t^{mn} \Big) - \sum_{t=0}^{T-1} \sum_{m \in M} q_m |\delta_t^m| \tag{14}$$

$$\text{subject to } z_t^m \le N_t^m \qquad \forall m \in M, t \in \mathcal{T} \tag{15}$$
$$\sum_{n \in N} c_n^r x_t^{mn} \le z_t^m C^{mr} \qquad \forall m \in M, r \in R, t \in \mathcal{T} \tag{16}$$
$$x_t^{mn}, z_t^m \in \mathbb{R}^+ \qquad \forall n \in N, m \in M, t \in \mathcal{T}$$

along with constraints (12) and (13). This problem is a convex optimization problem that can be solved using standard methods [8]. Before introducing the control algorithm, We provide some properties of CBS:

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

Proof: We rely on the property that the First-Fit (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 $x_t^{mn^*}$ 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 $\left\lfloor \frac{x_t^{mn*}}{2|R|} \right\rfloor$ 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 the fact that the total utilization is at most $\frac{z_t^{m^*}}{2|R|}$. ■

Lemma 1 provides a simple algorithm for rounding the fractional solution of $CBS-RELAX$ to an integer one using FF, such that at least $\left\lfloor \frac{x_t^{mn}}{2|R|} \right\rfloor$ containers of type $n \in N$ are scheduled on $z_t^{m*} + 1$ machines for each $m \in M$ and $n \in N$. Algorithm 1 summarizes our controller algorithm. When the control interval t starts, the controller uses the predicted values $N_{t+i|t}^n \forall n \in N, 1 \le i \le W$[2] to solve $CBS - RELAX$, 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 $\left\lfloor \frac{x_t^{mn}}{2|R|} \right\rfloor$ and then add containers using FF to ensure the number of type n containers is at least $\left\lfloor \frac{x_t^{mn*}}{2|R|} \right\rfloor$ for all $n \in N$. Container reassignment (i.e., migration) is then performed to ensure there are at most $z_{t|t}^{m*} + 1$ machines to be active. In our formulation, container reassignment cost is modeled as part of the machine switching cost, as it is only used to allow 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 containers for each $n \in N$ is at most x_t^{mn}. Finally, the controller will realize the

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

Algorithm 1 Controller Algorithm for CBS

Provide initial state z_0^m, x_0^{mn}, $t \leftarrow 0$
loop
  At beginning of control period t:
    Predict $N^n_{t+i|t}$, $p_{t+i|t}$ for horizons $t = 1, \cdots, W$ using a demand prediction model
    Solve $CBP - RELAX$ to obtain $\delta^m_{t+i|t}$,$\sigma^{mn}_{t+i|t}$ for $i = 0, \cdots, W-1$
    Sort new containers based on their utilities
    for $m \in M$ **do**
      Select $z^m_{t|t}$ machines of type m as active machines
    end for
    Compute a re-packing configuration for all selected active machines
    Turn on selected machines, perform re-parking, turn off other machines
    $t \leftarrow t + 1$
end loop

new configuration by actually turning off unused machines and making container allocations.

Theorem 1. *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 $CBS - RELAX$, 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^{mn*}, we have $E_t^{util} \leq E_t^{util*}$. Finally, by Lemma 1, it is easy to show that $\left\lfloor \frac{x_t^{mn*}}{2|R|} \cdot \frac{z_t^{m*}-1}{z_t^{m*}} \right\rfloor$ containers of each type $n \in N$ can be packed in z_t^{m*} machines. As $f(\cdot)$ is convex function, it must hold that $U_t^{pref} \geq (\max_m\{\frac{z_t^{m*}-1}{z_t^{m*}}\} - \epsilon') \cdot \frac{1}{2|R|} U_t^{pref*} \cdot$, where $\epsilon' = \max_m\{\frac{x_t^{mn*}}{2|R|} \cdot \frac{z_t^m-1}{z_t^m} - \left\lfloor \frac{x_t^{mn*}}{2|R|} \cdot \frac{z_t^m-1}{z_t^m} \right\rfloor\}$ 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 1 provides a bound on the worst case performance of CBS. In experiments, we have observed Algorithm 1 typically performs much better than the worst case bound. Furthermore, there are heuristics for finding better trade-offs between performance objectives U_t^{pref} and resource costs $(E_t + C_t^{sw})$. In particular, realizing the bin-packing solutions often cannot fully utilize the machine capacities, we can define an *over-provisioning factor* $\omega^n \in \mathbb{R}^+$ for container type n to account for the bin-packing inefficiencies. ω^n essentially captures how much extra resource is required to fully pack a given set of type n containers. To account for ω^m, it suffices to replace constraint (16) by the following constraint:

$$\sum_{n \in N} \omega^n c_n^r x_t^{mn} \leq z_t^m C^{mr} \quad \forall m \in M, r \in R, t \in \mathcal{T} \quad (17)$$

and run Algorithm 1 to find a suitable container placement. Finding a suitable value for ω^n can be done through various methods, such as uniformly sampling the range $[1, 2|R|]$ and selecting the one that produces the best solution among the sampled values for ω^n. However, using ω^n 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 $CBS-RELAX$ to be active. All other machines are inactive and have $E^{idle} \approx \infty$. In this case, no matter how we adjust the value of ω^n, the number of containers scheduled of the algorithm will not improve.

TABLE II: 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

VIII. Discussion

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

A. Task Classification

It is 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 pre-defined 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.

B. Scheduler Design

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 scheduler algorithms, implementing CBS requires major change to the scheduler design. To address this issue, in this section we propose a simple solution called *Contained-Based Provisioning (CBP)* that works with existing scheduling algorithms. Specifically, we solve $CBS - RELAX$ to compute the machines to be provisioned at run-time, and round the fractional values of δ_t^m and σ_t^{mn} to their nearest integer values as the number of machines available to the scheduler. At run-time, the scheduler can retain its current scheduling algorithm (e.g., first fit) as long as it ensures the number of type n tasks assigned to type m machines is less than x_t^{mn}. The main benefit of CBP is its simplicity and practicality for deployment in existing systems. However, due to lack of control of the scheduler, CBP does not provide performance guarantee in terms of task scheduling delay. As a solution that can be readily deployed in practice, we also evaluate the performance of CBP in our experiments.

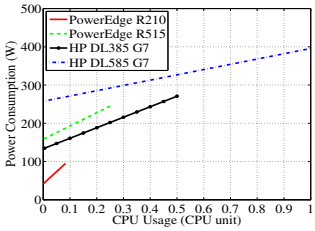


Fig. 9: Machine Energy consumption Rate

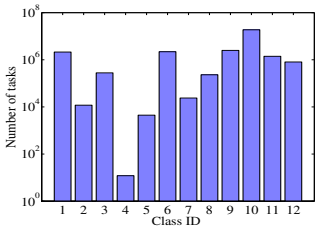


Fig. 10: Number of Tasks (gratis)

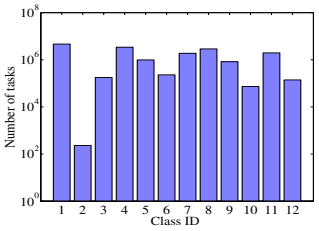


Fig. 11: Number of Tasks (Other)

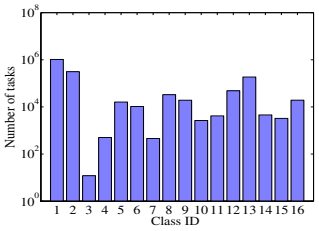


Fig. 12: Number of Tasks (Production)

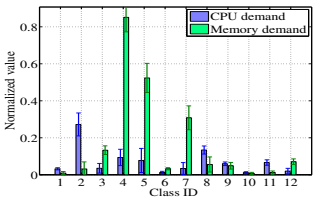


Fig. 13: Class size (Gratis)

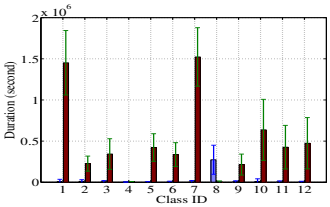


Fig. 14: Task duration (Gratis)

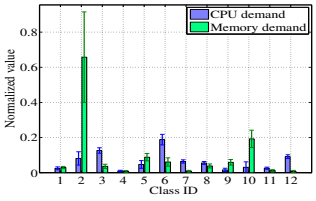


Fig. 15: Class size (Other)

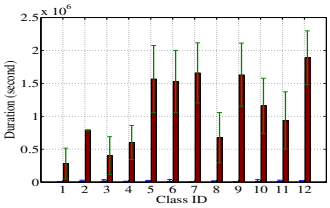


Fig. 16: Task duration (Other)

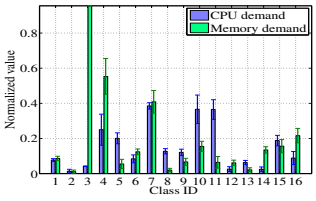


Fig. 17: Class size (Production)

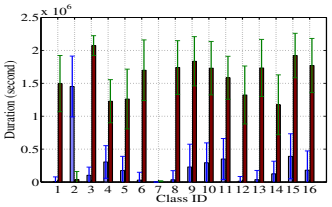


Fig. 18: Task duration (Production)

IX. Simulation Studies

We simulate a heterogeneous cluster composed of a mixture of servers from multiple manufacturers and models. Table II provides the characteristics of the simulated servers. We normalized the CPU core count 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. We used Equation 7 to model the energy consumption of the different machines. The parameters $E^{idle,m}$ and α^{mr} for each type of servers were estimated using energy measurements available in [2]. Figure 9 shows the energy consumption as function of CPU usage. Indeed, this figure demonstrates the importance of considering the machine heterogeneity when scheduling tasks/containers in order to reduce energy consumption. For instance, a container requiring 0.2 CPU unit should be placed in a HP DL385 G7 since the PowerEdge R210 does not have 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 even more challenging when millions of heterogeneous tasks have to be scheduled in the cluster.

A. Results for Task Classification

We performed task classification as described in Section V. 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 Figure 13, 15, and 17, respectively. It is evident that 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, confirming the accuracy of the characterization. The number of tasks in each task class is shown in Figure 10, 11 and 12, 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, 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 Figure 14, 16 and 18, respectively.

B. Controller Performance

We have evaluated the performance of CBS and CBP algorithms using Google workload traces. For comparison purpose, we have also implemented a baseline (heterogeneity-oblivious) algorithm that finds the best trade-off between energy savings and scheduling delay by maintaining an 80% utilization of the bottleneck resource. It provisions machines in a "greedy" fashion by turning them on in decreasing order of energy efficiency. As the Google workload contains many long running tasks that were scheduled before the start of the traces, in our current simulation, we mainly focus on simulating the arrival of new tasks. The sum of arrival rate of tasks belonging to each priority group is shown in Figure 19. Figure 20 shows the sum of the total containers belonging to each priority group computed by HARMONY. The number of active servers provisioned by the baseline algorithm and CBS/CBP are shown in Figure 21 and Figure 22, respectively. Note that CBS and CBP provision the same number of machines as

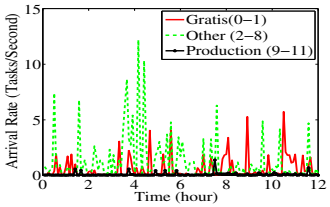


Fig. 19: Aggregated Task Arrival Rates

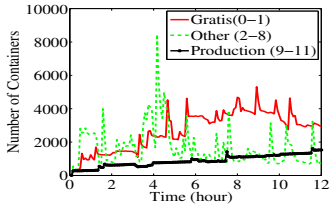


Fig. 20: Number of required containers

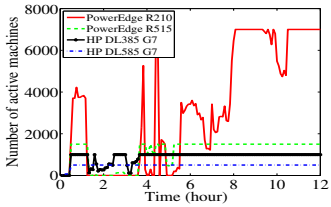


Fig. 21: Number of machines used by the baseline

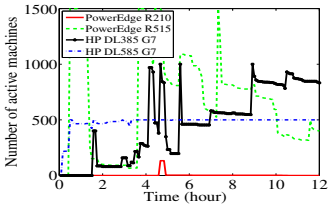


Fig. 22: Number of machines used by CBS and CBP

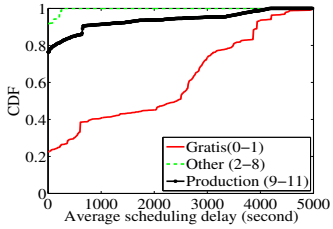


Fig. 23: CDF of scheduling delay for baseline

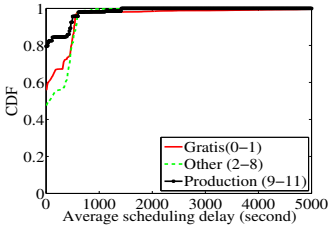


Fig. 24: CDF of scheduling delay for CBS

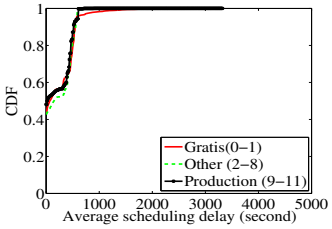


Fig. 25: CDF of scheduling delay for CBP

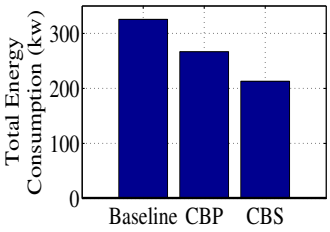


Fig. 26: Comparison of Energy Consumption

indicated by the MPC algorithm. The CDF of task scheduling delays are shown in Figure 23, 24 and 25, respectively. It can be seen that CBS can substantially reduce the scheduling delay compared to the baseline algorithm. On the other hand, CBP is able to outperform the baseline algorithm, but generally performs worse than CBS due to lack of coordination with the scheduling algorithm. An intuitive explanation is that the baseline algorithm was unable to take advantage of heterogenous machines CPU and memory capacities for scheduling, resulting in inefficient schedules and long scheduling delay for large tasks. Finally, Figure 26 shows the total energy consumption of all three approaches. It can be seen that CBS incurs the lowest energy costs, corresponding to a 28% reduction in energy cost compared to the baseline algorithm.

X. 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 both 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 can lead to up to 28 % energy savings while significantly improving task scheduling delay.

Acknowledgement

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/.
[2] Energy star computer server qualified product list. energystar.gov/ia/products/prod_lists/enterprise_servers_prod_list.xls.
[3] Googleclusterdata - traces of google workloads. http://code.google.com/p/googleclusterdata/.
[4] Notes on clustering. mlg.eng.cam.ac.uk/teaching/3f3/1011/lect4.pdf.
[5] Technology research - Gartner Inc. www.gartner.com.
[6] R. Boutaba, L. Cheng, and Q. Zhang. On cloud computational models and the heterogeneity challenge. *J. Internet Services and App*, 2012.
[7] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis, Forecasting, and Control*. Prentice-Hall, 1994.
[8] S. Boyd et al. *Convex optimization*. Cambridge University Press, 2004.
[9] C. Chekuri and S. Khanna. On multi-dimensional packing problems. In *Symposium on Discrete algorithms*, 1999.
[10] Y. Chen et al. Analysis and lessons from a publicly available Google cluster trace. *Tech. Rep. UCB/EECS-2010-95*, 2010.
[11] J. Diaz et al. A guide to concentration bounds. In *Handbook on Randomized Computing*, 2001.
[12] S. Ghemawat et al. The Google file system. In *SOSP*, 2003.
[13] D. Gross and C. Harris. Fundamentals of queueing theory. *ISBN: 0-471-17083-6*, pages 244–247, 1998.
[14] 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. In *ICDCS*, 2010.
[15] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das. Towards characterizing cloud backend workloads: insights from Google compute clusters. *SIGMETRICS Perform. Eval. Rev.*, 37, March 2010.
[16] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the electric bill for Internet-scale systems. In *ACM SIGCOMM*, 2009.
[17] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *ACM Symposium on Cloud Comp.*, 2012.
[18] S. Ren et al. Provably-efficient job scheduling for energy and fairness in geographically distributed data centers. In *ICDCS 2012*.
[19] B. Sharma et al. Modeling and synthesizing task placement constraints in google compute clusters. *ACM Symposium on Cloud Comp.*, 2011.
[20] A. Verma et al. pmapper: power and migration cost aware application placement in virtualized systems. In *Middleware*, 2008.
[21] Q. Zhang, J. Hellerstein, and R. Boutaba. Characterizing task usage shapes in Google's compute clusters. In *LADIS Workshop*, 2011.
[22] 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. of the IEEE/ACM International Conference on Autonomic Computing (ICAC)*, September 2012.