Dynamic Energy-Aware Capacity Provisioning for Cloud Computing Environments

Qi Zhang
University of Waterloo
Waterloo, ON, Canada
q8zhang@uwaterloo.ca

Mohamed Faten Zhani
University of Waterloo
Waterloo, ON, Canada
mfzhani@uwaterloo.ca

Shuo Zhang
National University of Defense Technology
Changsha, Hunan, China
zhangshuo@nudt.edu.cn

Quanyan Zhu
University of Illinois at Urbana-Champaign, USA
zhu31@illinois.edu

Raouf Boutaba
University of Waterloo
Waterloo, ON, Canada
Pohang University of Science and Technology (POSTECH)
Pohang 790-784, Korea
rboutaba@uwaterloo.ca

Joseph L. Hellerstein
Google, Inc.
Seattle, Washington, USA
jlh@google.com

ABSTRACT

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 initial capital investment over large number of machines, they also incur tremendous energy cost in terms of power distribution and cooling. An effective approach for saving energy in data centers is to adjust dynamically the data center capacity by turning off unused machines. However, this dynamic capacity provisioning problem is known to be challenging as it requires a careful understanding of the resource demand characteristics as well as considerations to various cost factors, including task scheduling delay, machine reconfiguration cost and electricity price fluctuation.

In this paper, we provide a control-theoretic solution to the dynamic capacity provisioning problem that minimizes the total energy cost while meeting the performance objective in terms of task scheduling delay. Specifically, we model this problem as a constrained discrete-time optimal control problem, and use *Model Predictive Control* (MPC) to find the optimal control policy. Through extensive analysis and simulation using real workload traces from Google's compute clusters, we show that our proposed framework can achieve significant reduction in energy cost, while maintaining an acceptable average scheduling delay for individual tasks.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; I.2.8 [**Problem Solving, Control Methods, and Search**]: Control Theory; I.6.3 [**Simulation and Modeling**]: Applications

General Terms

Management, Performance, Experimentation

Keywords

Cloud Computing; Resource Management; Energy Management; Model Predictive Control

1. INTRODUCTION

Data centers today are home to a vast number and a variety of applications with diverse resource demands and performance objectives. Typically, a cloud application can be divided into one or more tasks executed in one or more containers (e.g., virtual machines (VMs)). At run time, schedulers are responsible for assigning tasks to machines. In today's reality, production data centers such as Google's cloud backend often execute tremendous number (e.g., millions) of tasks on a daily basis [27]. Such extremely large-scale workload hosted by data centers not only consumes significant storage and computing power, but also huge amounts of energy. In practice, the operational expenditure on energy not only comes from running physical machines, but also from cooling down the entire data center. It has been reported that energy consumption accounts for more than 12% of monthly operational expenditures of a typical data center [5]. For large companies like Google, a 3% reduction in energy cost can translate into over a million dollars in cost savings [19]. On the other hand, governmental agencies continue to implement standards and regulations to promote energy-efficient (i.e., "Green") computing [1]. Motivated by these observations, cutting down electricity cost has become a primary concern of today's data center operators.

In the research literature, a large body of recent work tries to improve energy efficiency of data centers. A plethora of techniques have been proposed to tackle different aspects of the problem, including the control of power distribution systems [20], cooling systems [8], computer hardware [25], software components such as virtualization [24] and load-balancing algorithms [11, 15]. It is known that one of the most effective approach for reducing energy cost is to dynamically adjust the data center capacity by turning off unused machines, or to set them to a power-saving (e.g., “sleep”) state. This is supported by the evidence that an idle machine can consume as much as 60% of the power when the machine is fully utilized [11, 15, 17]. Unsurprisingly, a number of efforts are trying to leverage this fact to save energy using techniques such as VM consolidation [24] and migration [23]. However, these studies have mainly focused on improving the utilization of clusters by improving the “bin-packing” algorithm for VM scheduling. In a production data center where resource requests for tasks can arrive dynamically over time, deciding the number of machines to be switched off is not only affected by the efficiency of the scheduling algorithm, but also time-dependent characteristics of resource demand. While over-provisioning the data center capacity can lead to sub-optimal energy savings, under-provisioning the data center capacity can cause significant performance penalty in terms of scheduling delay, which is the time a task has to wait before it is scheduled on a machine. A high scheduling delay can significantly hurt the performance of some services that must be scheduled as soon as possible to satisfy end user requests (e.g., user-facing applications). On the other hand, tasks in production data centers often desire multiple types of resources, such as CPU, memory, disk and network bandwidth. In this context, devising a dynamic capacity provisioning mechanism that considers demand for multiple types of resources becomes a challenging problem. Furthermore, there are reconfiguration costs associated with switching on and off machines. In particular, turning on and off a machine often consumes large amount of energy due to saving and loading system states to memory and disk [18]. When turning off a machine with running tasks, it is necessary to consider the performance penalty due to migrating (or terminating) the tasks on the machine. Therefore, the reconfiguration cost due to server switching should be considered as well. Finally, another aspect often neglected in the existing literature is the electricity price. For example, it is known that in many regions of the U.S., the price of electricity can change depending on the time of the day. Electricity price is thus another factor that should be considered when making capacity adjustment decisions.

In this paper, we present a solution to the dynamic capacity provisioning problem with the goal of minimizing total energy cost while maintaining an acceptable task scheduling delay. Different from existing works on server capacity provisioning problem, we formulate the problem as a convex optimization problem that considers multiple resource types and fluctuating electricity prices. We then analyze the optimality condition of this problem and design a *Model Predictive Control* (MPC) algorithm that adjusts the number of servers to track the optimality condition while taking into account switching costs of machines. Through analysis and simulation using real workload traces from Google's compute clusters, we show our proposed solution is capable of achieving significant energy savings while minimizing SLA violations in terms of task scheduling delay.

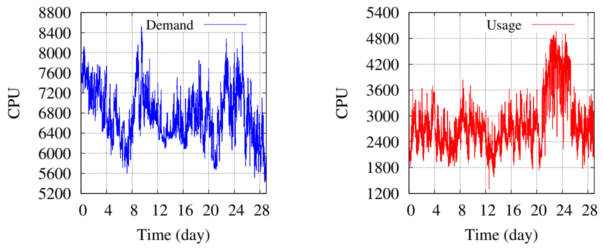
The remainder of the paper is organized as follows. Section 2 presents a survey of related work in the research literature. In Section 3, we present an analysis of real workload traces for one of Google's production compute clusters and illustrate the benefits of our approach. Section 4 describes the architecture of our proposed system. In Section 5, we present our demand prediction model and control algorithm. In Section 6, we provide our detailed formulation for the optimal control problem and present our solution based on the MPC framework. In Section 7, we evaluate our proposed system using Google workload traces, and demonstrate the benefits under various parameter settings. Finally, we draw our conclusions in Section 8.

2. RELATED WORK

Much effort has been made to achieve energy savings in data centers. Dynamic capacity provisioning is one of the most promising solutions to reduce energy cost that consists of dynamically turning on and off data center servers. For instance, motivated by the time-dependent variation of the number of users and TCP connections in Windows live messenger login servers, Chen et al. [11] have derived a framework for dynamic server provisioning and load dispatching. They have proposed a technique to evaluate the number of needed servers based on the predicted load in terms of users' login rate and active TCP connections. The load dispatching algorithm ensures that incoming requests are distributed among the servers. However, their framework does not consider the cost of switching on and off machines. Guenter et al. [16] have proposed another automated server provisioning and load dispatching system based on the predicted demand while considering the cost of transitioning servers between different power states (e.g., “on”, “off”, “hibernate”). This cost depends on the transition time, the energy cost and the long-term reliability of the server. Different from our work, they analyze the number of requests that can be satisfied instead of request scheduling delay. Furthermore, the multi-dimensional aspect of resource demand and fluctuations of electricity prices are not considered in their work.

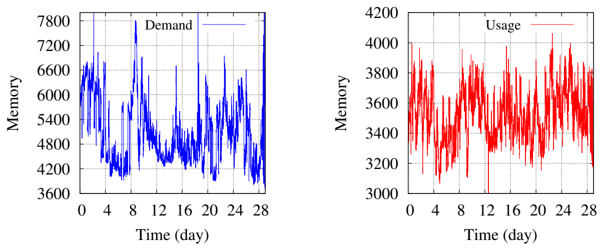
Kusic et al. [18] have proposed a dynamic resource provisioning framework for virtualized server environments based on lookahead control. The framework minimizes power consumption by adjusting the number of physical and virtual machines. It also estimates the CPU share and the workload directed to every virtual machine. In addition, their controller manages to maximize the number of transactions that satisfy Service Level Agreement (SLA) in terms of average response time while taking into account the cost of turning on and off the machines. However, they mainly consider the performance of application servers rather than the scheduling of VMs. Furthermore, time-dependent variations of electricity prices are not considered in their framework. Abbasi et al. [7] have proposed a thermal-aware server provisioning technique and a workload distribution algorithm. In this approach, active servers are selected using heuristics in a way that minimizes cooling and computing energy cost. The workload distribution algorithm ensures that servers' utilizations do not exceed a threshold in order to satisfy SLA defined in terms of average response time. However, their approach does not consider switching cost of machines.

There is also a large body of work that applies control theory to achieve energy savings in data centers. Fu et al. [15] have proposed a control-theoretic thermal balancing that reduces temperature differences among servers. Hence, the controller acts on the utilization of each processor in order to reduce or increase its temperature. Model predictive control



(a) Demand (b) Usage

Figure 1: Total CPU demand and usage (29 days).



(a) Demand (b) Usage

Figure 2: Total memory demand and usage (29 days).

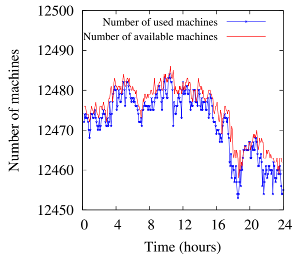


Figure 3: Number of machines available and used in the cluster.

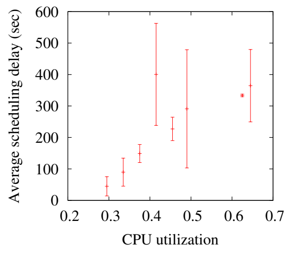


Figure 4: Average task scheduling delay vs. Resource utilization.

is used by Wang et al. [26] to reduce the total power consumption of a cluster by tuning CPU frequency level for the processor of each server. However, most of previous work has focused on capacity provisioning from a service provider perspective, i.e., provisioning server capacities (e.g., number of web servers) to accommodate end user requests. Existing solutions to this problem rely on queuing-theoretic models that consider only a single type of resource (mainly CPU). In contrast, our approach investigates the problem from the cloud provider's perspective, where resource demand and usage are multi-dimensional. Our solution considers resource usage and capacity for multiple resource types, such as CPU and memory. Furthermore, none of the existing work has considered additional factors such as the fluctuating electricity prices. Our approach is also lightweight and independent of the scheduling algorithm, making it more suitable for practical implementation.

3. WORKLOAD ANALYSIS

To motivate the problem and justify our solution approach, we have conducted an analysis of workload traces for a production compute cluster at Google [4] consisting of approximately $12,000$ machines. The dataset was released on November 29, 2011. 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, memory and disk size. At run time, the *usage* of a task measures the actual consumption of each type of resources. The current Google cluster traces provide task demand and usage for CPU, memory and disk [1]. The usage of each type of resource is reported at 5 minute intervals. Our current analysis mainly focuses on CPU and memory, as they are typically scarce compared to disk. However, we believe it is straightforward to extend our approach to consider other resources such as disk space.

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 (e.g., user-facing or batch). The priority determines the importance of each task. The task placement constraints specify additional scheduling constraints concerning the machine configurations, such as processor architecture of the physical machine [21]. To simplify, we do not consider the scheduling class, priority and task placement constraints in our model. The analysis of these factors is left for future work.

We first plot the total demand and usage for both CPU and memory over the entire duration. The results are shown in Figure 1 and 2 respectively. The total usage at a given time is computed by summing up the resource usage of all the running tasks at that time. On the other hand, the total demand at a given time is determined by total resource requirement by all the tasks in the system, including the tasks that are waiting to be scheduled. From Figure 1 and 2, it can be observed that both usage and demand for each type of resource 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. A machine is used if there is at least one task running on it. Figure 3 shows that the capacity of this cluster is not adjusted based on resource demand, as the number of used machines is almost equal to the number of available machines. Combining the observations from Figure 1, 2 and 3, it is evident that a large number of machines can be turned off to save energy. For instance, we estimated that a perfect energy saving schedule where the provisioned capacity exactly matches the current demand can achieve about 22% and 17% percent resource reduction for CPU and memory, respectively, compared to provisioning capacity according to the peak demand. This indicates that there is great potential for energy savings in this compute cluster using dynamic capacity provisioning.

However, 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. Classic queuing theory indicates that task scheduling delay

[1]Note that the values reported in Google cluster traces were normalized between 0 and 1.

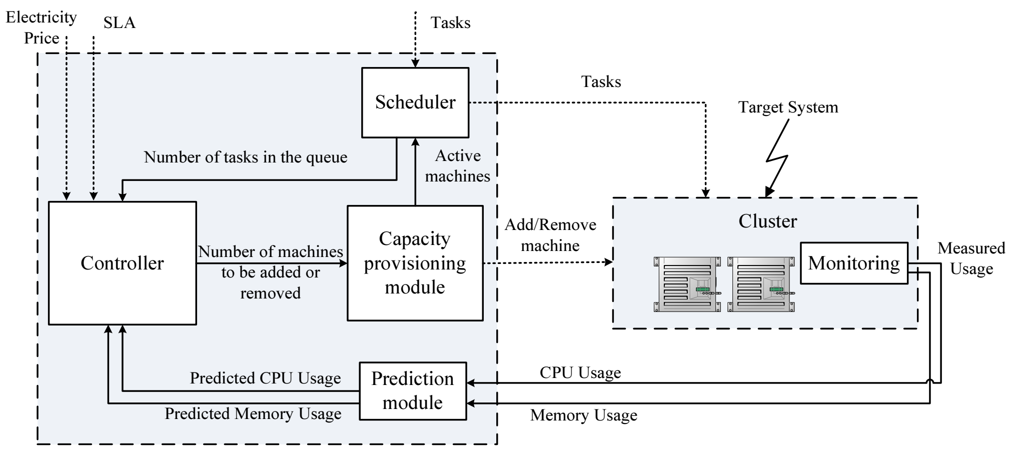


Figure 5: System architecture.

grows exponentially with resource utilization. To quantify this effect, we analyzed the relationship between scheduling delay experienced by each task and the average utilization of the bottleneck resource (e.g., CPU) while the task is waiting to be scheduled. We then plotted the average task scheduling delay as a function of the utilization of the bottleneck resource, as shown in Figure 4. The error bar in this diagram represents the standard deviation of task scheduling delay with average utilization at each given value. Indeed, we found that there is a direct relationship between task scheduling delay and resource utilization. We also modeled the relationship through curve fitting. It seems that both a linear function (i.e., $d = a \cdot U + b$) or a delay function for $M/M/1$ queuing model (i.e., $d = a \cdot \frac{U}{1-U} + b$)) can fit the curve well. Similar observations have been reported in recent work [27][21].

The above observations suggest that while the benefits of dynamic capacity provisioning is apparent for production data center environments, designing an effective dynamic capacity provisioning scheme is challenging, as it involves finding an optimal tradeoff between energy savings and scheduling delay. Furthermore, turning off active machines may require killing or migrating tasks running on these machines, which will introduce an additional performance penalty. The goal of this paper is to provide a solution to this dynamic capacity provisioning problem that finds the optimal trade-off between energy savings and the cost of reconfigurations, including cost of turning on and off machines and killing/migrating tasks.

4. SYSTEM ARCHITECTURE

Our proposed system architecture is depicted in Figure 5. It consists of the following components:

- *The scheduler* is responsible for assigning incoming tasks to active machines in the cluster. It also reports the average number of tasks in the queue during each control period to help the controller make informed decisions.

- *The monitoring module* is responsible for collecting CPU and memory usage statistics of every machine in the cluster. The monitoring is performed periodically.

- *The prediction module* receives statistics about the usage of all resources (CPU and memory) in the cluster and predicts the future usage for all of them.

- *The controller* implements a MPC algorithm that controls the number of machines based on the predicted usage of the cluster and taking into account the reconfiguration cost.

- *The capacity provisioning module* gathers the status information of machines from the controller, and decides which machines in particular should be added or removed. It then provides the scheduler with the list of active machines.

It is worth mentioning that different schedulers may adopt different resource allocation schemes. For example, in a public cloud environment such as Amazon EC2, it is necessary to schedule tasks according to their resource demand (e.g., VM size). However, since the actual resource usage of each task can be much lower than the demand, many advanced schedulers adjust dynamically resource allocation based on task usage [22]. Even though our framework is applicable to both scenarios, in this work, we use the latter case for illustration, not only because it is more general, but also because it reflects the behavior of Google cluster schedulers. In particular, Google's schedulers intentionally over-commit resources on each machine [3]. Finally, as an initial effort towards solving this problem, we currently consider that all the machines in the cluster are homogenous and with identical resource capacities. It is part of our future work to extend our model to consider machine heterogeneity (e.g., multiple generations of machines [21]).

In the following sections, we will describe the design of each component in details.

5. USAGE PREDICTION

In this section, we describe our model for predicting usage of each resource type. We used *the Auto-Regressive Integrated Moving Average* (ARIMA) model [9] to predict the time series G_k^r which represents the usage of resource type r in all the machines at time k. For convenience, we drop the superscript r and we write simply G_k in this section.

5.1 One-step Prediction

Knowing the last n observations of G_k, i.e., G_{k-n+1},... G_k, we want to predict G_{k+1}, which is the expected usage at time $k+1$ predicted at time k. The time series G_k follows an ARMA(n,q) model if it is stationary and if for every k:

$$\begin{aligned} G_{k+1} &= \phi_0 G_k + .. + \phi_{n-1} G_{k-n+1} \\ &\quad +\epsilon_{k+1} + \theta_0 \epsilon_k + .. + \theta_{q-1}\epsilon_{k+1-q}, \end{aligned} \quad (1)$$

where the ϕ_i and θ_j are constants estimated from available data. The terms ϵ_k are error terms which are assumed to be independent, identically distributed samples from a normal distribution with zero mean and finite variance σ^2. The parameters n and q are the number of lags used by the model (i.e., the number of last measured values of the usage) and the number of error terms respectively. Equation (1) can also be written in a concise form as:

$$G_{k+1} = \sum_{i=0}^{n-1} \phi_i L^i G_k + \epsilon_{k+1} + (\sum_{i=0}^{q-1} \theta_i L^i)\epsilon_k, \quad (2)$$

where L is the backward shift operator defined as follows: $L^i G_k = G_{k-i}$. We point out that AR and MA models are special cases of the ARMA model when $q = 0$ or $n = 0$.

The ARMA model fitting procedure assumes that the data are stationary. If the time series exhibits variations, we use differencing operation in order to make it stationary. It is defined by:

$$(1 - L)G_k = G_k - G_{k-1}. \quad (3)$$

It can be shown that a polynomial trend of degree k is reduced to a constant by differencing k times, that is, by applying the operator $(1-L)^k y(t)$. An ARIMA(n,d,q) model is an ARMA(n,q) model that has been differenced d times. Thus, the ARIMA(n,d,q) can be given by:

$$\left(1 - \sum_{i=0}^{n-1} \phi_i L^i\right)(1-L)^d G_k = \left(1 + \sum_{i=0}^{q-1} \theta_i L^i\right)\epsilon_k \quad (4)$$

5.2 Multi-step Prediction

In our model, we aim to predict future resource usage over a time window $H \in \mathbb{N}^+$. This requires predicting resource usage $h \in \mathbb{N}^+$ steps ahead from an end-of-sample G_k for all $1 \le h \le H$. Let $G_{k+h|k}$ denote the h^{th} step prediction of G_k knowing the last n observations, i.e., G_{k-n+1},... G_k. Thus, we aim to predict $G_{k+1|k}$,$G_{k+2|k}$....,$G_{k+h|k}$. The multi-step prediction is obtained by iterating the one-step ahead prediction. The h^{th} step prediction $G_{k+h|k}$ is given by:

$$G_{k+h|k} = f(G_{k+h-n|k}, ..., G_{k+h-i|k}, ..., G_{k+h-1|k}), \quad (5)$$

where $G_{k-i|k} = G_{k-i}\, \forall i \in [0, n]$, the function f is the prediction model, n is the number of lags used by the model and h is the prediction step. Table 1 illustrates how one-step prediction is iterated to obtain multi-step predictions.

6. CONTROLLER DESIGN

We formally describe the dynamic capacity provisioning problem in this section. We assume the cluster consists of $M_k \in \mathbb{N}^+$ homogeneous machines at time k. The number of machines in the cluster can change due to machine failure and recovery. We assume each machine has $d \in \mathbb{N}$ types of resources. For example, a physical machine provides CPU, memory and disk. Let $R = \{1, 2, ..., d\}$ denote the set of resource types. Denote by $C^r \in \mathbb{R}^+$ the capacity for resource type $r \in R$ of a single machine.

Table 1: Example of multi-step prediction ($n = 3$).

Prediction step	**Inputs of the model**	**Output**
1	$G_{k-2\|k}$,$G_{k-1\|k}$,$G_{k\|k}$	$G_{k+1\|k}$
2	$G_{k-1\|k}$,$G_{k\|k}$,$G_{k+1\|k}$	$G_{k+2\|k}$
3	$G_{k\|k}$,$G_{k+1\|k}$,$G_{k+2\|k}$	$G_{k+3\|k}$
4	$G_{k+1\|k}$,$G_{k+2\|k}$,$G_{k+3\|k}$	$G_{k+4\|k}$

To model the system dynamics, we divide time into intervals of equal duration. We assume reconfiguration happens at the beginning of each time interval. At interval $k \in \mathbb{N}^+$, the measured usage for resource type r in the cluster is denoted by G^r_k. Let x_k denote the number of active machines. Denote by $u_k \in \mathbb{R}$ the change in the number of active machines. A positive value of u_k means more machines will be turned on, whereas a negative value of u_k means some active machines will be powered off. Therefore, we have the following simple state equation that calculates the number of active machines at time $k + 1$:

$$x_{k+1} = x_k + u_k. \quad (6)$$

Our objective is to control the number of machines in order to reduce the total operational cost in terms of energy consumption and penalty due to violating the SLA, while taking into consideration the cost of dynamic reconfiguration. In what follows, we describe how to model each of the cost factors in details.

6.1 Modeling SLA penalty cost

In our model, the SLA is expressed in terms of an upper bound $\bar{d}$ on the average task scheduling delay. Thus, in order to meet the SLA, the average task scheduling delay d_k at time k should not exceed $\bar{d}$. As suggested in Section 3, the average task scheduling delay is correlated with the cluster resources' utilization, and more particularly with the utilization of the bottleneck resource. Therefore, we define the *bottleneck resource* $b \in R$ at time $k \in \mathbb{N}^+$ as the resource that has the highest utilization. In our model, the utilization of resource $r \in R$ in the cluster at time $k \in \mathbb{N}^+$ is given by:

$$U^r_k = \frac{G^r_k}{x_k C^r}. \quad (7)$$

Therefore, the utilization of the bottleneck resource b can be calculated as:

$$U^b_k = \max_{r \in R} \{U^r_k\}. \quad (8)$$

Then the average scheduling delay at time $k \in \mathbb{N}$ can be expressed as:

$$d_k = q_b(U^b_k), \quad (9)$$

where $q_b(U^b_k)$ denotes the average latency given current utilization U^b_k for the bottleneck resource b. The function $q_b(\cdot)$ can be obtained using various techniques, such as queue theoretic models, or directly from empirical measurements as described in Section 3.

We adopt a simple penalty cost model for SLA violation. Specifically, if the delay bound is violated, then there will be a SLA penalty cost P_k^{SLA} proportional to the degree of violation. Therefore, the penalty function $P_k^{SLA}(\cdot)$ can be rewritten as:

$$P_k^{SLA}(U_k^b) = N_k p^{SLA}(q(U_k^b) - \bar{d})^+, \tag{10}$$

where p^{SLA} represents the unit penalty cost for violating the delay upperbound, and N_k is the weight factor representing the severity of the violation at time k (e.g., N_k can be the number of requests in the scheduling queue at time k).

6.2 Modeling the total energy cost

In the research literature, it is known that the total energy consumption of a physical machine can be estimated by a linear function of CPU, memory and disk usage [16, 7, 11, 14]. Thus, the energy consumption of a machine at time k can be expressed as:

$$e_k = E_{idle} + \sum_{r \in R} \alpha^r U_k^r. \tag{11}$$

Let p_k^{power} denote the electricity price at time k. Then, for a given number of machines x_k, the total energy cost P_k^{power} at time k can be expressed as

$$\begin{aligned} P_k^{power}(x_k) &= p_k^{power} x_k e_k \\ &= p_k^{power} x_k (E_{idle} + \sum_{r \in R} \alpha^r \frac{G^r}{x_k C^r}). \end{aligned} \tag{12}$$

6.3 Formulation of the optimization problem

As mentioned previously, our objective is to control the number of servers so as to minimize the total operational cost, which is the sum of SLA penalty cost $P_k^{SLA}(U_k^b)$ and energy cost $P_k^{power}(x_k)$. At the same time, we need to ensure that the number of active machines in the cluster must not exceed M_k, the total number of physical machines in the cluster. This can be formulated by the following optimization problem:

$$\begin{aligned} \min_{x_k \in \mathbb{R}^+} \; & N_k p^{SLA} \left(q \left(\max_{r \in R} \left\{ \frac{G^r}{x_k C^r} \right\} \right) - \bar{d} \right)^+ \\ & + p_k^{power} x_k \left(E_{idle} + \sum_{r \in R} \alpha^r \frac{G^r}{x_k C^r} \right) \\ \text{s.t. } & 0 \le x_k \le M_k, \end{aligned} \tag{13}$$

where $(x)^+ = \max(x, 0)$. Notice that $p_k^{power} x_k \cdot \sum_{r \in R} \alpha^r \frac{G^r}{x_k C^r}$ does not depend on x_k at time k, thus it can be omitted in the optimization formulation. In addition, define $w_k = \max_r \{ \frac{G_k^r}{C^r} \}$, we can further simplify the problem to:

$$\min_{0 \le x_k \le M_k} N_k p^{SLA} \left(q \left(\frac{w_k}{x_k} \right) - \bar{d} \right)^+ + p_k^{power} x_k E_{idle}. \tag{14}$$

Assuming that $q(\frac{w_k}{x_k})$ is a decreasing function of x_k (namely, the average queuing delay decreases as the number of machines increases), we can see that the optimal solution x_k^* of this problem satisfies $x_k^* \le \frac{w_k}{q^{-1}(\bar{d})}$, since if $x_k^* \ge \frac{w_k}{q^{-1}(\bar{d})}$, $(q(\frac{w_k}{x_k}) - \bar{d})^+ = 0$, in this case we can decrease x_k^* to $\frac{w_k}{q^{-1}(\bar{d})}$ to reduce energy cost while maintaining $(q(\frac{w_k}{x_k}) - \bar{d})^+ = 0$. Therefore, we can further simplify the problem to:

$$\min_{0 \le x_k \le \frac{w_k}{q^{-1}(\bar{d})}} N_k p^{SLA}(q(\frac{w_k}{x_k}) - \bar{d}) + p_k^{power} x_k E_{idle} \tag{15}$$

In order to solve this optimization problem, we use the Karush-Kuhn-Tucker (KKT) approach [10]. The Lagrangian function is

$$\begin{aligned} L(x_k, \gamma) = & N_k p^{SLA} \left(q \left(\frac{w_k}{x_k} \right) - \bar{d} \right) + p_k^{power} x_k E_{idle} \\ & + \gamma(x_k - \frac{w_k}{q^{-1}(\bar{d})}) + \mu(0 - x_k). \end{aligned} \tag{16}$$

The KKT conditions are:

$$\begin{aligned} \frac{dL}{dx} = p_k^{power} E_{idle} - N_k p^{SLA} w_k \frac{dq\left(\frac{w_k}{x_k}\right)}{dx} \left(\frac{1}{x_k^2} \right) + \gamma - \mu = 0, \\ \mu x_k = 0, \\ \gamma(\frac{w_k}{q^{-1}(\bar{d})} - x_k) = 0, \\ 0 \le x_k \le \frac{w_k}{q^{-1}(\bar{d})}, \quad \mu, \gamma \ge 0. \end{aligned}$$

We need to consider three cases: (1) $\gamma > 0$, (2) $\mu > 0$, and (3) $\gamma = 0$ and $\mu = 0$. The first two cases correspond to boundary conditions whether $x_k^* = \frac{w_k}{q^{-1}(\bar{d})}$ or $x_k^* = 0$. In the third case, assuming $q(\cdot)$ is convex and differentiable, we can solve x_k^* using the first condition. For instance, $q(U) = a \cdot \frac{U}{1-U} + b$ (which is the case for $M/M/1$ queuing model), we can obtain

$$x_k^* \;=\; w_k + \sqrt{\frac{N_k p^{SLA} a w_k}{p_k^{power} E_{idle}}}. \tag{17}$$

The above equation reveals many insights. First, the optimal number of servers x_k^* depends mainly on the cluster utilization, which is captured by the variable w_k. Second, x_k^* is also dependent on the electricity price and the SLA violations. In particular, it increases either when the electricity price drops down or when the SLA penalty cost rises. Therefore, it can be seen that equation (17) tries to strike a balance between saving electricity cost and SLA penalty cost in a dynamic manner.

6.4 Capacity provisioning module

The capacity provisioning module takes as input the number of machines that should be added or removed from the cluster, and determines which machine should be turned on or off. The decision of switching on a particular machine can be made based on different criteria such as its usage and its location in the cluster. However, choosing which machine to power off is more complicated since some tasks could be running on it. Thus, more criteria should be considered such as the number of running tasks on the machine, their priorities, the cost of migrating or killing those tasks as well as the resource usage in the machine. For simplicity, define c_t as the cost for migrating (or terminating) the task t, depending on the scheduling policy applied to task t. For example, if task t is an interactive task such as a web server, it is better to migrate the server to another machine to minimize the service down time. On the other hand, if the task t belongs to a MapReduce job, it is more cost-effective to simply terminate the task and restart it on a different machine [12]. We define the cost of powering off a particular machine i, $1 \le i \le M_k$ at time k as

$$c_k^i = \sum_{t \in S_k^i} c_t, \tag{18}$$

Algorithm 1 MPC Algorithm for DCP

```
91: Provide initial state $x_k$, $k \leftarrow 0$
92: loop
93:   At the end of control period $k$:
94:     Predict $N_{k+h|k}$, $w_{k+h|k}$, $p^{power}_{k+h|k}$ for time $h \in \{1, H\}$
95:     Solve tracking problem to obtain $u(k+h|k)$ for horizons $h \in \{0, \cdots, H\}$
96:     Perform the reconfiguration using the capacity provisioning module according to $u(k|k)$
97:     $k \leftarrow k+1$
98: end loop
```

where S_k^i denotes the set of tasks running on the machine i at time $k \in \mathbb{N}$. It is clear that c_k^i increases with the number of tasks and their costs. Consequently, the capacity provisioning module turns off machines having the lowest cost c_k^i.

6.5 Designing the MPC controller

The goal of our controller is to adjust the number of machines to minimize the violation of KKT conditions, while taking into consideration the reconfiguration cost. As N_k, w_k, p_k^{power} can change over time, we adopt the well-known MPC framework to design an online controller for this problem. The MPC algorithm is illustrated by Algorithm 1. It can be intuitively described as follows: At time k, the prediction module is responsible for predicting the future values of N_k, w_k, p_k^{power} for a prediction window H. The controller will then solve an optimal control problem that will determine the optimal decisions for the entire window H. As only the first step is required, the controller will only carry out the first control decision. The procedure will repeat at the beginning of every time interval k, $k+1$, and so on.

More formally, we can define $N_{k+h|k}$,$w_{k+h|k}$, $p^{power}_{k+h|k}$ as the values of N_k, w_k, p_k^{power} predicted for time $k+h$, given the historical values up to time k. We also define

$$e_{k+h|k} = x_{k+h|k} - x^*_{k+h|k}, \tag{19}$$

as the *tracking error* at time k, the objective of the controller is to solve the following program:

$$\min_{u_k \in \mathbb{R}} J_k = \sum_{h=1}^{H} Q(e_{k+h|k})^2 + R(u_{k+h|k})^2 \tag{20}$$

$$\begin{aligned} \text{s. t. } \; & x_{k+h+1|k} = x_{k+h|k} + u_{k+h|k}, && \forall 0 \le h \le H-1 \\ & e_{k+h|k} = x_{k+h|k} - x^*_{k+h|k}, && \forall 1 \le h \le H \\ & 0 \le x_{k+h|k} \le N, && \forall 1 \le h \le H \end{aligned}$$

where H is the horizon of interest. The first term represents the tracking error, the second term represents the *control penalty*. The tracking error aims to reduce the error between the actual and the optimal number of machines. The second term is the control penalty which takes into account the cost of adding or removing machines. Thus, J_k can be interpreted as a *plan of action* for next H time intervals. Q and R are weight factors that will control the stability and convergence rate of the controller. If Q is much bigger than R, then the controller will place a higher weight on maximizing power savings and adjust number of servers aggressively. On the other hand, if R is large compared to Q, then the controller will adjust the capacity less aggressively to minimize reconfiguration cost. A standard way to determine the values of Q and R is to normalize both terms. In our case, we normalize them by converting both terms into monetary cost. Define $\bar{r} = \frac{1}{2}(c^{avg,on} + c^{avg,off})$ where $c^{avg,on}$ and $c^{avg,off}$ are the average cost for turning on and off servers, respectively. Similarly, we define $\bar{q} = \frac{1}{2}(c^{avg}_{over} + c^{avg}_{under})$ where c^{avg}_{over} is average cost introduced per machine due to provisioning, and c^{avg}_{under} is average cost introduced per machine due to underprovisining, respectively. Even though it is possible to compute analytically the values of $c^{avg,on}$, $c^{avg,off}$, c^{avg}_{over}, c^{avg}_{under}, it is more practical to estimate their values through empirical measurement. Notice that we set $\bar{q}$ and $\bar{r}$ to the average penalty cost of both positive and negative errors, because in practice, the number of occurrences of both positive and negative errors will likely to be the same, if the capacity provisioned by our controller only fluctuates within a fixed range. Finally, although we can set $(Q, R) = (\bar{q}^2, \bar{r}^2)$ to ensure both terms are in the unit of *dollar*^{2}, to simplify our model, we set $(Q, R) = (1, \frac{\bar{r}^2}{\bar{q}^2})$ in our experiment so that we only need to control R to achieve different trade-offs between solution optimality and reconfiguration cost.

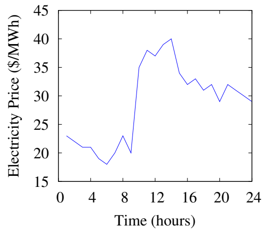


Figure 6: Electricity price of Houston, TX over 24 hours.

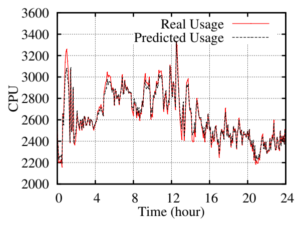
7. EXPERIMENTAL EVALUATION

We have implemented our system shown in Figure 5 and evaluated the quality of our solution using trace-driven simulations. In our experiment, we set the CPU and memory capacity of each machine to 1. This represents a majority of machines in the Google cluster[2]. In our simulation we implemented a greedy First-Fit (FF) scheduling algorithm, which is used by many cloud computing platforms such as Eucalyptus [2]. In our simulations, we set E_{idle} to 200 Watts, α^r to 121 and 0 for CPU and memory, respectively, similar to the values used in [18]. For electricity price, we used the electricity prices for the city of Houston, Texas, obtained from a publicly available source [6]. Figure 6 shows the fluctuation of electricity price over a duration of 24 hours. It can be observed that the electricity price is generally higher during day time. The fluctuation sometimes can be as large as 20% compared to the average electricity price over the 24 hours. Finally, we set $\bar{d}$ to 10 seconds as an upperbound on task scheduling delay.

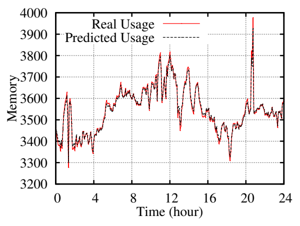
7.1 Prediction performance

In our first experiment, we assess the performance of the multi-step prediction for resource usages. We first describe the prediction procedure and the performance criteria. Then

[2]The values of CPU and memory capacity reported in Google traces were normalized to the configuration of the largest machine.

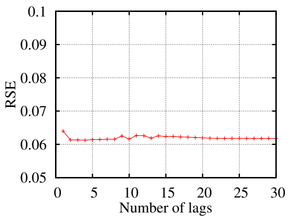


(a) CPU

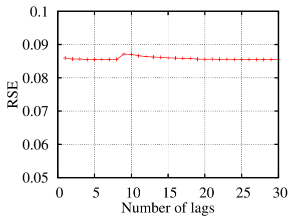


(b) Memory

Figure 7: Prediction of resource usage in the Google cluster - one-step prediction - ARIMA(2, 1, 1).



(a) CPU



(b) Memory

Figure 8: Effect of the number of lags on usage prediction - One-step prediction ($h = 1$).

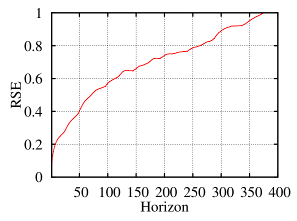
we study the effect of the number of lags and the prediction horizon on the prediction accuracy.

To evaluate the quality of our prediction technique, the available traces (e.g., measured CPU or memory usage) are divided into a *training data set* and a *validation data set*. Training data are used to identify the model parameters n, d, q and the coefficient ϕ_i and θ_j. For given n and q, the coefficients ϕ_i, $i \leq n$ and θ_j, $j \leq q$ are estimated using the RPS toolkit [13]. The validation data set is then used to assess the accuracy of the prediction model. The performance metric used to evaluate the prediction accuracy is *the relative squared error* (RSE). It is calculated for every prediction step h as:

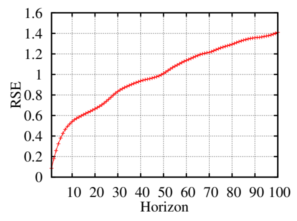
$$RSE_h = \frac{\sum_{k=1}^{T} \left[G_k - G_{k+h|k}\right]^2}{\sum_{k=1}^{T} \left[G_k - \mu\right]^2} \tag{21}$$

where T is the size of the validation data set and μ is the average of G_k over the T time intervals. The advantage of the RSE_h is that it neither depends on the used scale nor on the size of data. Having the RSE_h lower than 1 means that the predictor is outperforming the use of the average of the data as prediction for G_k ($G_{k+h|h} = \mu$). In addition, the smaller is the RSE_h , the more accurate is the prediction. The RSE can also be seen as the ratio of the mean squared error divided by the variance of validation data set.

Since our model exploits the predicted usage of the cluster in terms of CPU and memory to proactively add and remove servers, we assess the prediction model accuracy. We applied the ARIMA model to the real data collected at the Google cluster and we evaluated the effect of the number of lags used as input for the prediction model (n) and the effect of the prediction horizon (h) on the multi-step squared error (RSE_h). Memory and CPU usage are measured every five minutes. Hence, a one-step prediction is equivalent to predict the cluster usage in the next five minutes.



(a) CPU



(b) Memory

Figure 9: Performance of multi-step prediction - ARIMA(2, 1, 1).

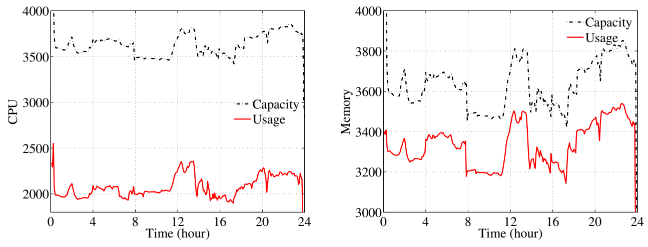
Figure 7 shows the one-step prediction of the CPU and memory usage compared to the real usage. The graph shows that the predicted values are always close to the real ones even during peaks. The prediction relative squared error RSE_1 is close to zero which proves that the ARIMA(2,1,1) provides an accurate prediction of the usage either for CPU ($RSE_1 \approx 0.062$) or memory ($RSE_1 \approx 0.086$).

Figure 8 depicts the effect of increasing the number of lags used in the ARIMA model to predict CPU and memory usage. Regarding CPU usage prediction (Figure 8(a)), it is apparent from the results that starting from the second lag ($n = 2$), the prediction error becomes stable around $RSE_1 \approx 0.062$. If we now turn to memory usage prediction, Figure 8(b) shows the prediction error remains almost stable regardless of the number of lags used for the ARIMA model ($RSE_1 \approx 0.086$). Consequently, there is no improvement of the prediction performance beyond two lags ($n = 2$). This result is interesting since a small number of lags reduces the ARIMA model complexity and allows to implement it online with minimal overhead and high accuracy.

We also conducted more experiments to examine the impact of the horizon h on the prediction performance. Since using more than two lags does not reduce the prediction error, we only considered two lags as input for the ARIMA model (i.e., $n = 2$). As expected, when we increase the prediction horizon, the prediction error grows for both CPU and memory usage (Figure 9). What is interesting in these results is that the error remains small ($RSE_h \leq 1$) for multiple prediction steps. In particular, the prediction error RSE_h remains below 1 for 400 steps ahead ($\approx$ 33 hour) for CPU usage and for 50 steps ahead ($\approx$ 250 min) for memory usage. We also mention that increasing the number of error terms (q) for the ARIMA model does not improve the prediction performance. In summary, these results suggest that we can apply ARIMA(2,1,1) using two lags to predict 12 steps ahead (equivalent to one hour), and this ensures that the prediction error does not exceed 0.3 and 0.5 for CPU and memory, respectively (Figure 9).

7.2 Controller performance

We conducted several experiments to evaluate the performance of our controller. In our experiment, we set the control frequency to once every 5 minutes to match Google's Cluster measurements frequency [4]. Typically, a high control frequency implies fast response to demand fluctuations. However, it also incurs a high computational overhead. However, we found the computational overhead of both demand prediction algorithm and controller to be almost negligible, thus, once every 5 minutes is a reasonable control frequency.



(a) CPU (b) Memory

Figure 10: Capacity vs. Usage over 24 hours (R=0.1).

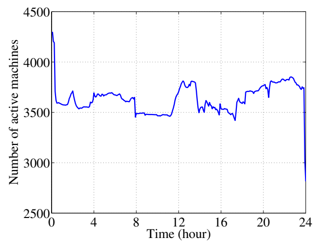


Figure 11: Number of machines over 24 hours (R=0.1).

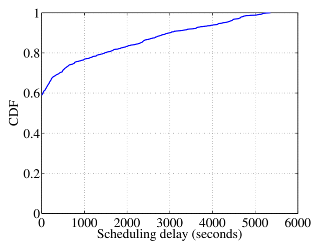


Figure 12: Average scheduling delay over 24 hours (R=0.1).

Lastly, we set $Q = 1$ in our experiments. Thus, the reconfiguration cost can be controlled by properly adjusting the value of R.

In our experiments, we first evaluated the response of our system to usage fluctuation. The number of active servers provisioned over the 24-hour duration is shown in Figure 11 (for $R = 0.1$). Figure 10 show the capacity and the usage of the cluster. It can be observed that the controller adjusts the number of servers dynamically in reaction to usage fluctuation, while avoiding rapid change in the number of active machines. The cumulative distribution function of task scheduling delay is shown in Figure 12. It can be seen that more than 60% of tasks are scheduled immediately.

We performed several other experiments for comparison purpose. In the first experiment, the number of machines is provisioned statically according to peak usage (i.e., 4100 machines). In the remaining experiments, we applied our controller using different values of R. Figures 13 and 14 show the corresponding average scheduling delay and energy consumption for different values of R compared to the static provisioning. It can be observed that the static provisioning achieves the lowest scheduling delay since it significantly overprovisions the cluster capacity. On the other hand, dynamic provisioning with $R = 0.5$ causes a significant scheduling delay although it allows to reduce the energy consumption (up to 50%). Furthermore, setting R to a small value (e.g., 0.02) does not achieve significant energy reduction. Through experiments, we found that setting $R = 0.225$ achieves our desired SLA objective of keeping the average scheduling delay around 10 seconds while reducing the energy consumption by 18.5%. Figure 15 shows the actual energy cost per hour, taking into consideration the fluctuation of the electricity price. It can be seen that our dynamic capacity provisioning mechanism reduces 7 dollars per hour

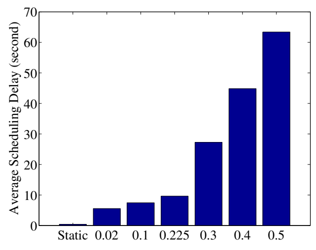


Figure 13: Average Queuing Delay.

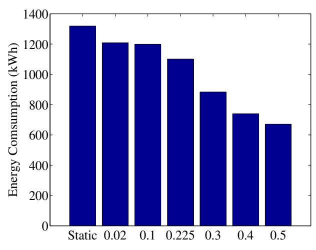


Figure 14: Energy Consumption.

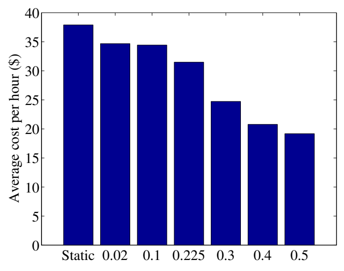


Figure 15: Average cost per hour.

in energy costs, which implies 20% reduction in energy cost, while achieving the desired scheduling delay (for $R = 0.225$). Furthermore, depending on the desired average scheduling delay (Figure 13), our proposed approach can reduce total operational cost by about $18.5 - 50\%$ (Figure 15).

8. CONCLUSION

Data centers have become a cost-effective infrastructure for data storage and hosting large-scale service applications. However, large data centers today consume significant amounts of energy. This not only rises the operational expenses of cloud providers, but also raises environmental concerns with regard to minimizing carbon footprint. In this paper, we mitigate this concern by designing a dynamic capacity provisioning system that controls the number of active servers in the data center according to (1) demand fluctuation, (2) variability in energy prices and (3) the cost of dynamic capacity reconfiguration. Our solution is based on the well-established Model Predictive Control framework, and aims to find a good trade-off between energy savings and capacity reconfiguration cost. Simulations using real traces obtained from a production Google compute clusters demonstrate our approach achieves considerable amount of reduction in energy cost. As such, we believe our approach represents an initial step towards building a full-fledged capacity management framework for cloud data centers.

There are several promising directions we can pursue in the future. First, our current approach assumes that machines are homogenous. While this is applicable to many situations (cloud providers often buy large quantities of identical machines in bulk), recent literature suggests that production data centers often consists of multiple types (sometimes multiple generations) of machines. Extending our current solution to handle machine heterogeneity requires careful consideration of scheduling capability of each type of machine. Another interesting problem is to understand the interplay between the scheduler and the capacity controller.

We believe it is possible to further reduce the cost of energy consumption and reconfiguration (i.e., task preemption and migration cost) if the scheduler and the capacity controller can cooperate tightly at a fine-grained level (e.g., interaction of server consolidation algorithms with our capacity controller).

Acknowledgment

This work was supported in part by a Google Research Award, by the Natural Science and Engineering Council of Canada (NSERC) SAVI Research Network, and by the World Class University (WCU) Program under the Korea Science and Engineering Foundation funded by the Ministry of Education, Science and Technology (Project No. R31-2008-000-10100-0).

9. REFERENCES

[1] Energy star computers specification - feb. 14, 2012. http://www.energystar.gov/ia/partners/prod_development/revisions/downloads/computer/ES_Computers_Draft_1_Version_6.0_Specification.pdf.

[2] Eucalyptus community. http://open.eucalyptus.com/.

[3] Google cluster-usage traces: format + schema. http://googleclusterdata.googlecode.com/files/Google%20cluster-usage%20traces%20-%20format%20%2B%20schema%20%282011.10.27%20external%29.pdf.

[4] Googleclusterdata - traces of google workloads. http://code.google.com/p/googleclusterdata/.

[5] Technology research - Gartner Inc. http://www.gartner.com/it/page.jsp?id=1442113.

[6] U.S. Energy Information Administration (EIA). http://www.eia.gov/.

[7] Z. Abbasi, G. Varsamopoulos, and S. K. S. Gupta. Thermal aware server provisioning and workload distribution for Internet data centers. In *Proceedings of the ACM International Symposium on High Performance Distributed Computing (HPDC)*, 2010.

[8] C. Bash, C. Patel, and R. Sharma. Dynamic thermal management of air cooled data centers. In *IEEE Intersociety Conference on the Thermal and Thermomechanical Phenomena in Electronics Systems (ITHERM)*, 2006.

[9] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis, Forecasting, and Control*. Prentice-Hall, third edition, 1994.

[10] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, USA, 2004.

[11] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive Internet services. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[12] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1), 2008.

[13] P. A. Dinda. Design, implementation, and performance of an extensible toolkit for resource prediction in distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 17, February 2006.

[14] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the annual international symposium on Computer architecture (ISCA)*, 2007.

[15] Y. Fu, C. Lu, and H. Wang. Robust control-theoretic thermal balancing for server clusters. In *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, April 2010.

[16] B. Guenter, N. Jain, and C. Williams. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *IEEE INFOCOM*, April 2011.

[17] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu. VMware distributed resource management: Design, implementation, and lessons learned. In *VMware Technical Journal*, 2012.

[18] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, 2008.

[19] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the electric bill for Internet-scale systems. In *ACM SIGCOMM Computer Communication Review*, volume 39, 2009.

[20] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGARCH Computer Architecture News*, volume 36. ACM, 2008.

[21] B. Sharma, V. Chudnovsky, J. Hellerstein, R. Rifaat, and C. Das. Modeling and synthesizing task placement constraints in google compute clusters. In *Proceedings of ACM Symposium on Cloud Computing*, 2011.

[22] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the ACM Symposium on Cloud Computing*, 2011.

[23] A. Verma, P. Ahuja, and A. Neogi. pMapper: power and migration cost aware application placement in virtualized systems. In *ACM/IFIP/USENIX Middleware*, 2008.

[24] A. Verma, G. Dasgupta, T. Nayak, P. De, and R. Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the conference on USENIX Annual technical conference*. USENIX Association, 2009.

[25] G. von Laszewski, L. Wang, A. Younge, and X. He. Power-aware scheduling of virtual machines in DVFS-enabled clusters. In *IEEE International Conference on Cluster Computing and Workshops (CLUSTER)*, 2009.

[26] X. Wang and M. Chen. Cluster-level feedback power control for performance optimization. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, February 2008.

[27] Q. Zhang, J. Hellerstein, and R. Boutaba. Characterizing task usage shapes in Google's compute clusters. In *Workshop on Large Scale Distributed Systems and Middleware (LADIS)*, 2011.