Energy-Optimized Dynamic Deferral of Workload for Capacity Provisioning in Data Centers

Muhammad Abdullah Adnan*, Ryo Sugihara†, Yan Ma‡ and Rajesh K. Gupta*
*University of California San Diego, †Amazon.com, ‡Shandong University

Abstract—This paper explores the opportunity for energy cost saving in data centers that utilizes the flexibility from the Service Level Agreements (SLAs) and proposes a novel approach for capacity provisioning under bounded latency requirements of the workload. We investigate how many servers to keep active and how much workload to delay for energy saving while meeting latency constraints. We present an offline LP formulation for capacity provisioning by dynamic deferral and give two online algorithms to determine the capacity of the data center and the assignment of workload to servers dynamically. We prove the feasibility of the online algorithms and show that their worst case performances are bounded by constant factors with respect to the offline formulation. To the best of our knowledge, this is the first formulation for capacity provisioning in data centers considering workload deferral with bounded latency. We validate our algorithms on MapReduce workload by provisioning capacity on a Hadoop cluster and show that the algorithms actually perform much better in practice compared to the naive ‘follow the workload’ provisioning, resulting in 20-40% cost-savings.

I. INTRODUCTION

With the advent of cloud computing, data centers are emerging all over the world and their energy consumption becomes significant; as estimated 61 million MWh per year, costing about 4.5 billion US dollars [1]. Naturally, energy efficiency in data centers has been pursued in various ways including the use of renewable energy [2], [3] and improved cooling efficiency [4], [5], etc. Among these, improved scheduling algorithm is a promising approach for its broad applicability regardless of hardware configurations. Among the attempts to improve scheduling [5], [6], [7], recent effort has focused on optimization of schedule under performance constraints imposed by Service Level Agreements (SLAs). Typically, a SLA specification provides a measure of flexibility in scheduling that can be exploited to improve performance and efficiency [8], [9]. To be specific, latency is an important performance metric for any web-based service and is of great interest for service providers who run their services on data centers. The goal of this paper is to utilize the flexibility from the SLAs for different types of workload to reduce energy consumption. The idea of utilizing SLA information to improve performance and efficiency is not new. Recent work explores utilization of application deadline information for improving the performance of the applications (e.g. see [9], [10]). But the opportunities for energy efficiency remain unexplored, certainly in a manner that seeks to establish bounds on energy cost from the proposed solutions.

In this paper, we are interested in minimizing the energy consumption of a data center under guarantees on average latency or deadline. We use the latency (deadline/average latency) information to defer some tasks so that we can reduce the total cost for energy consumption for executing the workload and switching the state of the servers. We determine the portion of the released workload to be executed at the current time and the portions to be deferred to be executed at later time slots without violating latency constraints. Our approach is similar to ‘valley filling’ that is widely used in data centers to utilize server capacity during the periods of low loads [6]. But the load that is used for valley filling is mostly background/maintenance tasks (e.g. web indexing, data backup) which is different from actual workload. In fact current valley filling approaches ignore the workload characteristics for capacity provisioning. In this paper, we determine how much work to defer for valley filling in order to reduce the current and future energy consumption while provably ensuring satisfaction of SLA requirements. Later we generalize our approach for more general workloads where different jobs have different latency requirements.

This paper makes three contributions. First, we present an LP formulation for capacity provisioning with dynamic deferral of workload. The formulation not only determines capacity but also determines the assignment of workload for each time slot. As a result the utilization of each server can be determined easily and resources can be allocated accordingly. Therefore this method well adapts to other scheduling policies that take into account dynamic resource allocation, priority aware scheduling, etc.

Second, we design two optimization based online algorithms depending on the nature of the latency requirement. For uniform requirement (e.g. all the jobs have same deadline), our algorithm named Valley Filling with Workload (VFW(δ)), looks ahead δ slots to optimize the total energy consumption. The algorithm uses the valley filling approach to defer some workload to execute in the periods of low loads. For nonuniform deadline, we design a Generalized Capacity Provisioning (GCP) algorithm that reduces the switching (on/off) of servers by balancing the workloads in adjacent time slots and thus reduces energy consumption. We prove the feasibility of the solutions and show that the performance of the online algorithms are bounded by a constant factor with respect to the offline formulation. To the best of our knowledge, this is the first algorithm for capacity provisioning in data centers considering workload deferral with bounded latency.

Third, we validate our algorithms using MapReduce traces (representative workload for data centers) and evaluate cost savings achieved via dynamic deferral. We run simulations to deal with a wide range of settings and show significant savings in each of them. Over a period of 24 hours, we find more than 40% total cost saving for GCP and around 20% total cost saving for VFW(δ) even for small deadline requirements. We compare the two online algorithms with different parameter settings and find that GCP gives more cost savings than VFW(δ). In order to show that our algorithms work on real systems, we perform experiments on a 35-node Hadoop cluster and find energy savings of ~6.02% for VFW(δ) and ~12% for GCP over a period of 4 hours. The experimental results show that the peak energy consumption for the operation of a data center can be reduced by provisioning capacity and scheduling workload using our algorithms.

The rest of the paper is organized as follows. Section II presents the model that we use to formulate the optimization and gives the offline formulation considering hard deadline requirements for the jobs. In Section III, we present the VFW(δ) algorithm for determining capacity and workload assignment dynamically when the deadline (latency requirement) is uniform. In Section IV, we illustrate the GCP algorithm with
nonuniform deadline. In Section V and VI, we illustrate the simulation and experimental results respectively. In Section VII, we extend our formulation for general latency requirements (soft deadlines) for the jobs. Section VIII describes the related work and Section IX concludes the paper.

II. MODEL FORMULATION

In this section, we describe the model we use for capacity provisioning via dynamic deferral.

A. Workload Traces

To build a realistic model, we need real workload from data centers but the data center providers are reluctant to publish their production traces due to privacy issues and competitive concerns. To overcome the scarcity of publicly available traces, efforts have been made to extract summary statistics from production traces and workload generators based on those statistics have been proposed [1], [2]. For the purposes of this paper, we use such a workload generator and use the MapReduce traces released by Chen et al [11]. MapReduce framework is widely used in Data centers and acts as representative workload where each of the jobs consists of 3 steps of computation: map, shuffle and reduce [14]. Figure 1(a) illustrates the statistical MapReduce trace over 24 hours generated from real Facebook traces.

Typically the workload traces consist of a mix of batch and interactive jobs. Chen et al. carried out an interactive analysis to classify the workload and showed that the workload is dominated (~98%) by small and interactive jobs showing significant and unpredictable variation with time. Table I illustrates the classification on the MapReduce traces by k-means clustering based on the sizes of map, shuffle and reduce stages (in bytes) with k = 10 and Figure 1(b) shows the distinction in time variation between the long batch jobs and small interactive jobs. To adapt with the large variation in the small and interactive workload, valley filling methods have been proposed using the low priority batch jobs to fill in the periods of low workload [13]. However, Chen et al. have shown that the portion of low priority long jobs (~2%) is insufficient to reduce the variation (to smooth) in the workload curve [12]. In this paper, we propose valley filling with workload (mix of long and interactive jobs) and devise algorithms for capacity provisioning by scheduling jobs under bounded latency requirements.

B. Workload Model

The workload model is over a time frame \( t \in \{0, 1, \ldots, T\} \) where \( T \) can be arbitrarily large. In practice, \( T \) can be a year and the length of a time slot \( \tau \) can be as small as 2 minutes (the minimum time required to change power state of a server). Let \( L_t \) be the amount of workload released at time slot \( t \). The workload \( L_t \) can contain short jobs and long jobs. If the length \( \ell \) of a job is greater than \( \tau \), we decompose the job into small pieces \( \leq \tau \) each of which is released after the execution of the preceding piece. Thus long jobs are decomposed into small jobs. Hence we do not distinguish each job, rather deal with the total amount of workload. Due to page limitation, we omit details of the length estimation and decomposition procedure in this paper; the details can be found in a technical report [15].

In our model, jobs have latency requirements specified in the SLAs. The latency requirements are specified in terms of hard/soft deadlines or average latency of completion. In the rest of this paper, we consider hard deadline requirements for the jobs. However, our model and algorithms can be extended for general latency requirements as discussed in Section VII. So, each job has a deadline \( D \) (in terms of number of slots) associated with it, where \( D \) is a nonnegative integer. A job released at time \( t \) needs to be executed within time slot \( t + D \). The value of \( D \) can be zero for interactive jobs and large for batch-like jobs. If the job is long and decomposed into smaller pieces, then we need to assign deadline to each individual piece. If the long job is preemptive then we assign deadline \( \lceil D/t \rceil - 1 \) to each of the small pieces and for a non-preemptive job, we assign deadline of \( D - \ell \) to the first piece and deadlines of zeros to the other pieces. To simplify analysis, we first consider the case of uniform deadlines, that is, deadline is uniform for all the jobs, followed by non-uniform deadline case in Section IV.

Since the deadline \( D \) is uniform for all the jobs, the total amount of work \( L_t \) must be executed by the end of time slot \( t + D \). Since \( L_t \) varies over time, we often refer to it as a workload curve.

We consider data center as a collection of homogeneous servers. The total number of servers \( M \) is fixed and given but each server can be turned on/off to execute the workload. We normalize \( L_t \) by the processing capability of each server i.e. \( \frac{L_t}{M} \) denotes the number of servers required to execute the workload at time \( t \). We assume for all \( t, L_t \leq M \). Let \( x_{i,d,t} \) be the portion of the released workload \( L_t \) that is assigned to be executed at server \( i \) at time slot \( t + d \) where \( d \) represents the deferral with \( 0 \leq d \leq D \). Let \( n_t \) be the number of active servers during time slot \( t \). Then \( \sum_{i=1}^{n_t} \sum_{d=0}^{D} x_{i,d,t} = L_t \) and \( 0 \leq x_{i,d,t} \leq 1 \).

Let \( x_{i,t} \) be the total workload assigned at time \( t \) to server \( i \) and \( x \) be the total assignment at time \( t \). Then we can think of \( x_{i,t} \) as the utilization of the \( i \)th server at time \( t \) i.e. \( 0 \leq x_{i,t} \leq 1 \). Thus \( \sum_{i=0}^{D} x_{i,t} = x \) and \( \sum_{i=0}^{n_t} x_{i,t} = x_t \). From the data center perspective, we focus on two important decisions during each time slot \( t \): (i) determining \( n_t \), the number of active servers, and (ii) determining \( x_{i,d,t} \), assignment of workload to the servers.

C. Cost Model

The goal of this paper is to minimize the cost of energy consumption in data centers. The energy cost function consists of two parts: operating cost and switching cost. Operating cost is the cost for executing the workload which in our model is proportional to the assigned workload. We use the common model for energy cost for typical servers which is an affine

---

**TABLE I**

| # Jobs | % Jobs | Input | Shuffle | Output |
|-------|-------|-------|---------|--------|
| 5691  | 96.56 | 15 KB | 0       | 685 KB |
| 116   | 1.97  | 44 GB | 15 GB   | 84 MB  |
| 27    | 0.46  | 56 GB | 145 GB  | 16 GB  |
| 23    | 0.39  | 123 GB| 0       | 52 MB  |
| 19    | 0.32  | 339 KB| 0       | 48 GB  |
| 8     | 0.14  | 203 GB| 404 GB  | 3 GB   |
| 5     | 0.08  | 529 GB| 0       | 53 KB  |
| 3     | 0.05  | 46 KB | 0       | 199 GB |
| 1     | 0.02  | 7 TB  | 48 GB   | 101 GB |
| 1     | 0.02  | 913 GB| 8 TB    | 61 KB  |

---

**Fig. 1.** Illustration of (a) original workload and (b) distinction between batch and small interactive jobs.
function:  
\[ C(x) = e_0 + e_1 x \]

where \( e_0 \) and \( e_1 \) are constants (e.g. see [16]) and \( x \) is the assigned workload (utilization) of a server at a time slot. Although we use this general model for cost function, other models considering nonlinear parameters such as temperature, frequency can easily be adopted in the model which will make it a nonlinear optimization problem. Our algorithms can be applied for such nonlinear models by using techniques for solving nonlinear optimizations as each algorithm is considered as a single independent step in the algorithms.

Switching cost \( \bar{\beta} \) is the cost incurred for changing state (on/off) of a server. We consider the cost of both turning on and turning off a server. Switching cost at time \( t \) is defined as follows:

\[ S_t = \bar{\beta}|m_t - m_{t-1}| \]

where \( \bar{\beta} \) is a constant (e.g. see [6], [17]).

D. Optimization Problem

Given the models above, the goal of a data center is to choose the number of active servers (capacity) \( m_t \) and the dispatching rule \( x_{i,d,t} \) to minimize the total cost during \([1, T]\), which is captured by the following optimization:

\[
\min_{x_t, m_t} \quad T \sum_{t=1}^{T} \sum_{i=1}^{m_t} C(x_{i,t}) + \bar{\beta} \sum_{t=1}^{T} |m_t - m_{t-1}| \\
\text{subject to} \quad \sum_{i=1}^{m_t} D_i x_{i,d,t} = L_t \quad \forall t \nonumber \\
\sum_{i=1}^{m_t} x_{i,d,t} \leq m_t \leq M \quad \forall t \nonumber \\
\sum_{d=0}^{T} x_{i,d,t-d} \leq m_t \leq M \quad \forall t \nonumber \\
\sum_{d=0}^{T} x_{i,d,t} \geq 0 \quad \forall i, \forall d, \forall t. 
\]  

(1)

Since the servers are identical, we can simplify the problem by dropping the index \( i \) for \( x \). More specifically, for any feasible solution \( x_{i,d,t} \), we can make another solution by \( x_{i,d,t} = \frac{m_t}{m_t} \times x_{i,d,t} \) (i.e., replacing every \( x_{i,d,t} \) by the average of \( x_{i,d,t} \) for all \( i \)) without changing the value of the objective function while satisfying all the constraints after this conversion. Then we have the following optimization equivalent to (1):

\[
\min_{x_t, m_t} \quad T \sum_{t=1}^{T} m_t C(x_{t}/m_t) + \bar{\beta} \sum_{t=1}^{T} |m_t - m_{t-1}| \\
\text{subject to} \quad \sum_{d=0}^{T} x_{d,t} = L_t \quad \forall t \nonumber \\
\sum_{d=0}^{T} x_{d,t-d} \leq m_t \leq M \quad \forall t \nonumber \\
\sum_{d=0}^{T} x_{d,t} \geq 0 \quad \forall d, \forall t. 
\]  

(2)

where \( x_{d,t} \) represents the portion of the workload \( L_t \) to be executed at active servers at time \( t + d \). We further simplify the problem by showing that any optimal assignment for (2) can be converted to an equivalent assignment that uses earliest deadline first (EDF) policy (see Figure 2). More formally, we have the following lemma.

**Lemma 1:** Let \( x^*_t \) and \( x^*_t \) be the optimal assignments of workload obtained from the solution of optimization (2) at times \( t_r \) and \( t_r \) respectively where \( t_s > t_r \) and \( t_s - t_r = \theta < D \). If \( \theta \) with \( \sum_{d=0}^{\delta} x_{d,t_0} \neq 0 \) and \( \sum_{d=\delta+1}^{D} x_{d,t_0-d} \neq 0 \) for any \( 0 < \delta < (D - \theta) \) then we can obtain another assignments \( x^*_t = x^*_t \) and \( x^*_t = x^*_t \) where \( \sum_{d=\delta}^{D} x_{d,t_0-d} = 0 \) and \( \sum_{d=\delta+1}^{D} x_{d,t_0-d} = 0 \).

**Proof:** We prove it by constructing \( x^*_t \) and \( x^*_t \) from \( x^*_t \) and \( x^*_t \). We change the assignments \( x^*_t \) and \( x^*_t \), \( 0 \leq \theta < (D - \theta) \) and \( x^*_t, \theta \leq \delta \leq D \) to obtain \( x^*_t \) and \( x^*_t \) by swapping some of the jobs without violating respective deadlines, as illustrated in Figure 2. We now determine \( \delta \). Note that all the jobs released between (including) time slots \( t_s - D \) to \( t_r \) can be executed at time \( t_r \) without violating deadline since \( t_s - D < t_r - \theta < t_r \). Also all the jobs released between (including) time slots \( t_s - D \) to \( t_r \) can be executed at time \( t_r \) without violating deadline since \( t_s - D < t_r - \theta < t_r \). Hence the new assignment of workloads cannot violate any deadline. We determine \( \delta \) at a point where \( \sum_{d=0}^{\delta} x_{d,t_0} = \sum_{d=\delta+1}^{D} x_{d,t_0-d} = 0 \) and \( \sum_{d=\delta+1}^{D} x_{d,t_0-d} = 0 \) and \( \sum_{d=\delta+1}^{D} x_{d,t_0-d} = 0 \)

According to lemma 1, we do not need both \( t \) and \( d \) as indices for \( x \). We can use the release time \( t \) to determine the deadline \( t + d \) and differentiate between jobs using their deadlines. Thus, we drop the index \( d \) of \( x \). At time \( t \) unscheduled workload from \( L_{t-D} \) to \( L_t \) is executed according to EDF policy while minimizing the objective function. To formulate the constraint that no assignment violates any deadline we define delayed workload \( l_t \) where all the jobs are delayed to their maximum deadline \( D \).

\[
l_t = \begin{cases} 
0 & \text{if } t \leq D, \\
L_{t-D} & \text{otherwise}. 
\end{cases}
\]

We call the delayed curve \( l_t \) for the workload as deadline curve. Thus we have two fundamental constraints on the assignment of workload for all \( t \):

(C1) Deadline Constraint: \( \sum_{j=1}^{t} l_j \leq \sum_{j=1}^{t} x_j \)

(C2) Release Constraint: \( \sum_{j=1}^{t} x_j \leq \sum_{j=1}^{t} l_j \)

Condition (C1) says that all the workloads assigned up to time \( t \) cannot violate deadline and Condition (C2) says that the assigned workload up to time \( t \) cannot be greater than the total released workload up to time \( t \). Using these constraints we reformulate the optimization (2) as follows:
\[
\min_{x_t, m_t} \sum_{t=1}^{T} m_t C(x_t/m_t) + \beta \sum_{t=1}^{T} |m_t - m_{t-1}|
\]  
subject to \[
\sum_{j=1}^{t} l_j \leq \sum_{j=1}^{t} x_j \leq \sum_{j=1}^{t} L_j \quad \forall t
\]
\[
\sum_{j=1}^{t} x_j = \sum_{j=1}^{t} L_j 
\]
\[
0 \leq x_t \leq m_t \leq M \quad \forall t
\]

Since the operating cost function \( C(\cdot) \) is an affine function, the objective function is linear as well as the constraints. Hence optimization (3) is a linear program. Note that capacity \( m_t \) in this formulation is not constrained to be an integer. This is acceptable because data centers consist of thousands of active servers and we can round the resulting solution with minimal increase in cost. Figure 3(a) illustrates the offline optimal solutions for \( x_t \) and \( m_t \) for a dynamic workload generated randomly. The performance of the optimal offline solution on two realistic workloads are provided in Section V.

### III. Valley Filling with Workload

In this section, we consider the online case, where at any time \( t \), we do not have information about the future workload \( L_{t'} \) for \( t' > t \). At each time \( t \), we determine the \( x_t \) and \( m_t \) by applying optimization over the already released unassigned workload which has deadline in future \( D \) slots. Note that the workload released at or before \( t \), can not be delayed to be assigned after time slot \( t + D \). Hence we do not optimize over more than \( D + 1 \) slots. We simplify the online optimization by solving only for \( m_t \) and determine \( x_t \) by making \( x_t = m_t \) at time \( t \). This makes the online algorithm to not waste execution capacity that cannot be used later. But the cost due to switching in the online algorithm may be higher than the offline algorithm as \( m_t \) can be larger than \( x_t \) in the offline formulation. Thus our goal is to design strategies to reduce the switching cost. In the online algorithm, we reduce the switching cost by optimizing the total cost for the interval \([t, t + D]\).

When the deadline is uniform, we can reduce the switching cost even more by looking beyond \( D \) slots. We do that by accumulating some workload from periods of high loads and execute that amount of workload later in valleys without violating constraints (C1) and (C2). Note that by accumulation we do not violate deadline as at each slot, we execute a portion of the accumulated workload by swapping with the newly released workload by EDF policy. To determine the amount of accumulation and execution we use ‘\( \delta \)-delayed workload’. Thus the online algorithm namely Valley Filling with Workload (VFW(\( \delta \))) looks ahead \( \delta \) slots to determine the amount of execution. Let \( l^\delta_t \) be the \( \delta \)-delayed curve with delay of \( \delta \) slots 

\[
I:\delta \rightarrow \{0\text{ if } t \leq \delta, L_{t-\delta}\text{ otherwise.}\}
\]

Then we can call the deadline curve as \( D\)-delayed curve and represent it by \( l^D_t \). We determine the amount of accumulation and execution by controlling the set of feasible choices for \( m_t \) in the optimization. For this purpose, we use the \( \delta \)-delayed curve to restrict the amount of accumulation. By having a lower bound on \( m_t \) for the valley (low workload) and an upper bound for the high workload, we control the execution in the valley and accumulation in the other parts of the curve. Thus in the online algorithm, we have two types of optimizations: Local Optimization and Valley Optimization. Local Optimization is used to smooth the ‘wrinkles’ (we define wrinkles as the small variation in the workload in adjacent slots e.g. see Figure 4) within \( D \) consecutive slots and accumulate some workload. On the other hand, Valley Optimization fills the valleys with the accumulated workload.

#### A. Local Optimization

The local optimization applies optimization over future \( D \) slots and finds the optimum capacity for current slot by executing no more than \( \delta \)-delayed workload. Let \( t \) be the current time slot. At this slot we apply a slightly modified version of offline optimization (3) in the interval \([t, t + D]\). We apply the following optimization \( \text{LOPT}(l_t, l^\delta_t, m_{t-1}, M) \) to determine \( m_t \) in order to smooth the wrinkles by optimizing over \( D \) consecutive slots. We restrict the amount of execution to be no more than the \( \delta \)-delayed workload while satisfying the deadline constraint (C1).

\[
\min_{m_t} \ (e_0 + e_1) \sum_{j=t}^{t+D} m_j + \beta \sum_{j=t}^{t+D} |m_j - m_{j-1}|
\]  
subject to \[
\sum_{j=1}^{t} l^D_j \leq \sum_{j=1}^{t} m_j 
\]
\[
t + D \leq t \leq k + D
\]
\[
0 \leq m_k \leq M \quad t \leq k \leq t + D
\]

After solving the local optimization, we get the value of \( m_t \) for the current time slot and assign \( x_t = m_t \). For the next time slot \( t + 1 \) we solve the local optimization again to find the values for \( x_{t+1} \) and \( m_{t+1} \). Note that the deadline constraint (C1) and the release constraint (C2) are satisfied at time \( t \), since from the formulation \( \sum_{j=1}^{t} l^D_j \leq \sum_{j=1}^{t} m_j \leq \sum_{j=1}^{t} l^\delta_j \leq \sum_{j=1}^{t} L_j \).

Fig. 3. Illustration of (a) offline optimal solution and (b) VFW(\( \delta \)) for arbitrary workload generated randomly; time slot length = 2 minute, \( D = 15, \delta = 10 \).

Fig. 4. The curves \( L_t \) and \( l^\delta_t \) and their intersection points. The peak from the \( l^\delta_t \) curve is cut and used to fill the valley of the same curve. The amount of workload that is accumulated/delayed is bounded by \( m_t D \).
B. Valley Optimization

In valley optimization, the accumulated workload from the local optimization is executed in ‘global valleys’. Before giving the formulation for the valley optimization we need to detect a valley.

Let $p_1, p_2, \ldots, p_n$ be the sequence of intersection points of $L_t$ and $L^d_t$ curves (see Figure 4) in nondecreasing order of their x-coordinates (t values). Let $p_1, p_2', \ldots, p_n'$ be the sequence of points on $L^d_t$ with delay $\delta$ added with each intersection point $p_1, p_2', \ldots, p_n'$. We discard all the intersection points (if any) between $p_s$ and $p_l'$ from the sequence such that $t_{s+1} = t'$. Note that at each intersection point $p_s$, the curve from $p_s$ to $p_l'$ is known. To determine whether the curve $L^d_t$ between $p_s$ and $p_l'$ is a valley, we calculate the area $A = \sum_{t=s}^{t'} (L^d_t - L_t)$. If $A$ is negative, then we regard the curve between $p_s$ and $p_l'$ as a global valley though it may contain several small peaks and valleys. If the curve between $p_s$ and $p_l'$ is a global valley, we fill the valley with the valley from $p_s$ such that $\delta = \sum_{t=s}^{t'} (L^d_t - L_t)$. If $A > 0$, we define the valley as a global valley and $\delta$ as the total cost by $\delta = \sum_{t=s}^{t'} (L^d_t - L_t)$.

Algorithm 1 VFW($\delta$)

1. valley ← 0; $m_0 ← 0$
2. $I^D[1 : D] ← 0$; $L^d[1 : \delta] ← 0$
3. for each new time slot $t$
4. \quad $I^D[t + D] ← L^d[t]$
5. \quad $L^d[t + \delta] ← L^d[t]$
6. if valley = 0 and $I^d_t$ intersects $L$
7. \quad Calculate Area $A = \sum_{t=s}^{t'} (L^d_t - L_t)$
8. \quad if $A < 0$ then
9. \quad \quad valley ← 1
10. \quad end if
11. else if valley > 0 and valley $\leq \delta$ then
12. \quad valley ← valley + 1
13. else
14. \quad valley ← 0
15. end if
16. if valley = 0 then
17. \quad $m[t : t + D] ← \text{LOPT}([1 : t], [t : t + D], m_{t-1}, M)$
18. else
19. \quad $m[t : t + D] ← \text{VOPT}([1 : t], [t : t + D], m_{t-1}, M)$
20. end if
21. $x_t ← m_t$
22. end for

We now consider the general case where deadline requirements are not same for all the jobs in a workload. Let $\nu$ be the maximum possible deadline. We decompose the workload according to their associated deadline. Suppose $L_{d,t,d} ≥ 0$ be the portion of the workload released at time $t$ and has deadline $d$, $0 ≤ d ≤ \nu$. We have $\sum_{d=0}^{\nu} L_{d,t,d} = L_t$. The workload to be executed at any time slot $t$ can come from different previous slots $t - d$ where $0 ≤ d ≤ \nu$ as illustrated in Figure 5(a). Hence we redefine the deadline curve $L_t$ and represent it by $L^d_t$. Assuming $L_{d,t,d} = 0$ if $t ≤ 0$, we define $L^d_t = \sum_{d=0}^{\nu} L_{d,t-\nu}$. Then the offline formulation remains the same as formulation.
(3) with the deadline curve $l_t$ replaced by $l'_t$.

$$\min_{x_t, m_t} \sum_{t=1}^{T} m_t C(x_t/m_t) + \beta \sum_{t=1}^{T} |m_t - m_{t-1}|$$  \hspace{1cm} (6)

subject to

$$\sum_{j=1}^{t} l'_j \leq \sum_{j=1}^{t} x_j \leq \sum_{j=1}^{T} L_j \quad \forall t$$

$$\sum_{j=1}^{T} x_j = \sum_{j=1}^{T} L_j$$

$$0 \leq x_t \leq m_t \leq M \quad \forall t$$

We now consider the online case. Delaying the workload up to their maximum deadline may increase the switching cost since it may increase the variation in the workload compared to the original workload (see Figure 5(b)). Hence at each time we need to determine the optimum assignment and capacity that reduces the switching cost from the original workload while satisfying each individual deadline. We can apply the VFW(δ) algorithm from the previous section with $D = D_{\text{min}}$ where $D_{\text{min}}$ is the minimum deadline for the workload. If $D_{\text{min}}$ is small, VFW(δ) does not work well because $\delta < D_{\text{min}}$ becomes too small to detect a valley. Hence we use a novel approach for distributing the workload $L_t$ over the $D_t$ slots such that the change in the capacity between adjacent time slots is minimal (see Figure 5(c)). We call this algorithm as Generalized Capacity Provisioning (GCP) algorithm.

![Fig. 5. Illustration of workload with different deadline requirements. (a) workload released at different times have different deadlines, (b) the delayed workload $l'_t$ may increase the switching cost due to large variation, (c) distribution of workload in adjacent slots by GCP to reduce the variation in workload.](image)

In the GCP algorithm, we apply optimization to determine $m_t$ at each time slot $t$ and make $x_t = m_t$. The optimization is applied over the interval $[t, t + \nu]$ since at time slot $t$ we can have workload that has deadline up to $t + \nu$ slots. Hence at each time $t$, the released workload is a vector of $\nu + 1$ dimension. Let, $L_t = (L_{0,t}, L_{1,t}, \ldots, L_{\nu,t})$ where $L_{d,t} = 0$ if there is no workload with deadline $d$ at time $t$. Let $y'_t$ be the vector of unassigned workload released up to time $t$. The vector $y_t$ is updated from $y'_{t-1}$ at each time slot by subtracting the capacity $m_{t-1}$ and then adding $L_t$. Note that $m_{t-1}$ is subtracted from the vector $y'_{t-1}$ in order to use unused capacity to execute already released workload at time $t - 1$ by following EDF policy (see lines 4-17 in Algorithm 2). Let $y'_t = (y'_{0,t-1}, y'_{1,t-1}, y'_{2,t-1}, \ldots, y'_{\nu,t-1})$ be the vector after subtracting $m_{t-1}$ with $y'_{d,t-1} = 0$ and $y'_{d,t-1} \geq 0$ for $1 \leq j \leq \nu$. Then $y_t = L_t + (y'_{0,t-1}, y'_{1,t-1}, y'_{2,t-1}, \ldots, y'_{\nu,t-1}, 0)$ where $y_t = (0, 0, \ldots, 0)$ if $t < 0$. Then the optimization GCP-OPT($y_t$, $m_{t-1}$, $M$) applied at each $t$ over the interval $[t, t + \nu]$ is as follows:

$$\min_{y_t} (c_0 + c_1) \sum_{j=t}^{t+\nu} m_j + \beta \sum_{j=t}^{t+\nu} |m_j - m_{j-1}|$$  \hspace{1cm} (7a)

subject to

$$\sum_{j=0}^{\nu} m_{j+1} = \sum_{j=0}^{\nu} y_{j,t}$$  \hspace{1cm} (7b)

$$\sum_{k=0}^{j} m_{k+1} \geq \sum_{k=0}^{j} y_{k,t} \quad 0 \leq j \leq \nu - 1$$  \hspace{1cm} (7c)

$$0 \leq m_{t+1} \leq M \quad 0 \leq j \leq \nu$$  \hspace{1cm} (7d)

Note that the optimization (7) solves for $\nu + 1$ values. We only use $m_t$ as the capacity and assignment of workload at time $t$. Algorithm 2 summarizes the procedures for GCP. The GCP algorithm gives feasible solutions because it works with the unassigned workload and constraint (7c) ensures deadline constraint (C1) and constraint (7b) ensures the release constraint (C2). The competitive ratio for the GCP algorithm is same as the competitive ratio for VFW(δ) because in GCP, $m_t = x_t$ and release constraint (C2) holds at every $t$ making $\sum_{t=1}^{T} m_t = \sum_{t=1}^{T} x_t \leq \sum_{t=1}^{T} L_t$.

![Algorithm 2 GCP](image)

V. SIMULATION

In this section, we evaluate the cost incurred by the VFW(δ) and GCP algorithm relative to optimal solution in the context of workload generated from realistic data. First, we motivate our evaluation by a detailed analysis of simulation results. Then in Section VI, we validate the simulation results by performing experiments on a Hadoop cluster.

A. Simulation Setup

We use realistic parameters in the simulation setup and provide conservative estimates of cost savings resulting from our proposed VFW(δ) and GCP algorithms.

Cost benchmark: A common approach for capacity provisioning in data centers is to follow the workload curve [6]. Such an approach is naive and does not take into account switching costs. Yet this is a conservative estimate as it does not waste any execution capacity and meets all the deadlines. We compare the total cost from the VFW(δ) and GCP algorithms with the ‘follow the workload’ ($x = m = L$) strategy and evaluate the cost reduction.
Workload B

in the experiments.

Fig. 6. Illustration of the two MapReduce traces as dynamic workload used in the experiments.

Cost function parameters: The total cost is characterized by $e_0$ and $e_1$ for the operating cost and $\beta$ for the switching cost. In the operating cost, $e_0$ represents the proportion of the fixed cost and $e_1$ represents the load dependent energy consumption. The energy consumption of the current servers is dominated by the fixed cost [18]. Therefore we choose $e_0 = 1$ and $e_1 = 0$. The switching cost parameter $\beta$ represents the energy wasted during switching, service migration overhead and wear-and-tear due to changing power states in the servers. We choose $\beta = 12$ for slot length of 5 minutes such that it works as an estimate of the time a server should be powered down (typically one hour [6], [17]) to outweigh the switching cost with respect to the operating cost.

Workload description: We use two publicly available MapReduce traces as examples of dynamic workload. The MapReduce traces were released by Chen et al. [11] which are produced from real Facebook traces for one day (24 hours) from a cluster of 600 machines. We count the number of different types of job submissions over a time slot length of 5 minutes and use that as a dynamic workload (Figure 6) for simulation. The two samples we use represent strong diurnal properties and have variation from typical workload (Workload A) to bursty workload (Workload B).

Deadline assignment: For VFW($\delta$), the deadline $D$ is uniform and is assigned in terms of number of slots the workload can be delayed. For our simulation, we vary $D$ from 1 – 12 slots which gives latency from 5 minutes upto 1 hour. This is realistic as deadlines of 8-30 minutes for MapReduce workload have been used in the literature [9], [23]. For GCP, we use k-means clustering to classify the workload into 10 groups based on the map, shuffle and reduce bytes. The characteristics of each group are depicted in Table II. From Table II, it is evident that smaller jobs dominate the workload mix, as discussed in Section IIA. For each class of jobs we assign a deadline from 1 – 10 slots such that smaller jobs have smaller deadlines and larger jobs have larger deadlines.

B. Analysis of the Simulation

We now analyze the impact of different parameters on cost savings provided by VFW($\delta$) and GCP. We then compare VFW($\delta$) and GCP for uniform deadline (GCP-U).

Impact of deadline: The first parameter we study is the impact of different deadline requirements of the workload on the cost savings. Figure 7 shows that even for deadline $D$ as small as 2 slots, the cost is reduced by $\sim 40\%$ for GCP-U. $\sim 20\%$ for VFW($\delta$) while the offline algorithm gives a cost saving of $\sim 60\%$ compared to the ‘follow the workload’ algorithm. It also shows that for all the algorithms, large $D$ gives more cost savings as more workload can be delayed to reduce the variation in the workload. As $D$ grows larger the cost reduction from GCP-U and VFW($\delta$) approaches offline cost saving which is as much as $70\%$. The cost savings from VFW($\delta$) is always less than GCP-U for both the workload.

Impact of $\delta$ for VFW($\delta$): The parameter $\delta$ is used as a lookahead to detect a valley in the VFW($\delta$) algorithm. If $\delta$ is large, valley detection performs well but it may be too late to fill the valley due to the deadlines. On the other hand if $\delta$ is small, valley detection does not work well because the capacity has already gone down to the lowest value. Figure 8 illustrates the valley detection for small $\delta$ and large $\delta$. Although the cost savings from VFW($\delta$) largely depends on the nature of the workload curve, Figure 9 shows that $\delta = D/2$ is a conservative estimate for better cost savings.

Performance of GCP: We evaluated the cost savings from GCP by assigning different deadline by classifying the workload as shown in Table II. For conservative estimates of deadline requirements (1-10), we found 47.66% cost reduction for Workload A and 45.65% cost reduction for Workload B each of which remains close to the offline optimal solutions.

Comparison of VFW($\delta$) and GCP: We compare GCP for uniform deadline (GCP-U) with VFW($\delta$) for $\delta = D/2$. Figure 7 illustrates the cost reduction for VFW($\delta$) and GCP-U with different deadlines $D = 1 – 12$. For both the workload, GCP-U performs better than VFW($\delta$). This is because GCP has more flexibility on deferral as opposed to following a fixed $\delta$-delayed curve. However for workloads with significant variability (peaks/valleys), valley filling with workload as in VFW($\delta$) can be more beneficial than provisioning capacity for $D$ consecutive slots as in GCP. Hence we conclude that the comparative performance of the online algorithms depends largely on the nature of the workload. Since both the algorithms are based on linear program, they take around 10-12 milliseconds to compute schedule at each step.
VI. EXPERIMENTATION

In this section, we validate our algorithms on MapReduce workload by provisioning capacity on a Hadoop cluster. We evaluate the cost-savings by energy consumption calculated from common power model using different measured metrics.

A. Experimental Setup

We setup a Hadoop cluster (version 0.20.205) consisting of 35 nodes on Amazon’s Elastic Compute Cloud (EC2) [19], [20]. Each node in the cluster is a small instance with 1 virtual core, 1.7 GB memory, 160 GB storage. We configured one node as master and four core nodes to contain the Hadoop DFS and the other 30 nodes as task nodes. The provisioning is done on the task nodes dynamically. We used the Amazon Elastic MapReduce service for provisioning capacity on the Hadoop cluster. The online algorithm is implemented in a central server (load balancer) outside of Hadoop cluster. The load balancer releases the jobs and provisions the capacity of the cluster according to the algorithm. Being elastic, Amazon Elastic MapReduce takes care of provisioning machines and migration of tasks between machines while keeping all data available. Moreover, as the number of servers and jobs are represented by variables, scalability is not an issue for the load balancer.

To generate the MapReduce workload for our cluster, we used the Statistical Workload Injector for MapReduce (SWIM) [11] using the Facebook traces from Figure 6(a). We run our experiment for 4 hours with slot length of 5 minutes. For the traces of Figure 6(a), 602 jobs were released in the first 48 slots.

We first schedule the jobs and provision the task nodes by the ‘follow the workload’ strategy. We then schedule the same jobs and provision the task nodes using our algorithms as illustrated in Figure 10. In order to make comparison between VF/f(α) and GCP algorithms we used a uniform deadline of 10 minutes (D = 2). In each of the experiments, we measured the seven metrics (available from Amazon Cloudwatch) for each of the ‘running’ nodes in each time slot over the time interval of 4 hours and 10 minutes (50 slots). In the last 2 slots, the capacity of the task nodes were provisioned to zero for the ‘follow the workload’ algorithm while our algorithms execute the delayed workload in those slots. All the jobs released in the first 48 slots were completed before the end of 50th slot. The seven metrics that are available for measurement for each virtual machine are: CPU Utilization, Disk Read Bytes, Disk Read Ops, Disk Write Bytes, Disk Write Ops, Network In and Network Out.

B. Experimental Results

We now discuss the results from the experimentation and compare energy consumption between different algorithms.

Power Measurement: We use the general power model [21], [22] to evaluate energy consumption for the algorithms. The energy consumed by a virtual machine is represented as the

\[
E_{vm}(t) = \alpha_{cpu} u_{cpu}(t) + \gamma_{cpu} + \alpha_{disk} d_{disk}(t) + \gamma_{disk} + \alpha_{dops} u_{dops}(t) + \gamma_{dops} + \alpha_{net} u_{net}(t) + \gamma_{net}
\]

where \(u_{cpu}(t)\) is the average utilization, \(b_{r}(t)\) and \(b_{w}(t)\) are the total bytes read and written to disk, \(n_{r}(t)\) and \(n_{w}(t)\) are the total number of disk read and writes and \(b_{in}(t)\) and \(b_{out}(t)\) are the total bytes of network I/O for the virtual machine over the time interval \(t\). Since the difference in energies for disk read and write and network input and output are negligible [21], we use common parameters \(b_{db}(t)\), \(b_{dw}(t)\), and \(b_{net}(t)\) by taking the sum of the respective values. We normalize these values with their respective maximum values (in the interval \(T\)) so that each of these become a fraction between 0 and 1 and can be put in equation (8),

\[
E_{vm}(t) = \alpha_{cpu} u_{cpu}(t) + \gamma_{cpu} + \alpha_{disk} d_{disk}(t) + \gamma_{disk} + \alpha_{dops} u_{dops}(t) + \gamma_{dops} + \alpha_{net} u_{net}(t) + \gamma_{net}
\]

where \(u_{disk}(t)\), \(u_{dops}(t)\), and \(u_{net}(t)\) represent the normalized values of \(b_{db}(t)\), \(b_{dw}(t)\), and \(b_{net}(t)\) respectively. If \(m\) machines are active at time slot \(t\), then the total energy consumed over the interval \(T\) can be computed using this equation:

\[
E(T) = \sum_{i=1}^{T} \frac{E_i(t) \times \frac{T}{3600}}{m}
\]

where \(E_i(t)\) is the energy consumed at machine \(i\) over time slot \(t\). To compute energy consumption, we used parameters from [22] listed in Table III. Typical values are used for cpu utilization, disk I/O and network I/O. Idle disk/network powers are negligible with respect to dynamic power and scale of workload.

The total energy consumption and the % reduction with respect to ‘follow the workload’ in each of the metrics for
different schedules are illustrated in Table IV. For the period of 4 hours 10 minutes (50 slots), GCP algorithm gives energy reduction of ∼12% which is significantly better than the reduction of ∼6.02% from the VFW(δ) algorithm. The reductions from both the algorithms are far better (more than 50%) with respect to workload schedule without provisioning. Table IV also shows that variation in CPU utilization, Disk I/O and Network I/O across different algorithms. This variation results from the difference in capacity provisioning across algorithms that changes migration of jobs and disk I/O in the cluster. Figure 11 illustrates the average energy consumption within each slot over the time interval showing significant reduction in the peak energy consumption. As the provisioning algorithms cut off peaks from the workload and provision the machines without wasting computation capacity, they reduce the peak energy consumption for the data center.

Choice of deadline, D: We choose the deadline requirement to be 10 minutes for the MapReduce workload. This is realistic because MapReduce workloads have deadlines in the range of minutes as deadlines from 8-30 minutes for these workloads have been used in the literature [8], [9], [23], [24]. Moreover deadline may vary for different applications and for some of them the deadline can be zero. Our GCP algorithm is designed considering all these cases and it works well for any kind of deadline/workload mix as demonstrated by simulation and experiment. Again, the deadline requirement for some applications e.g. High Performance Computing (HPC) is large in order of hours and for some applications it is small in order of minutes [24], [25]. Hence we study the impact of different deadline requirements varying from 10 minutes to 1 hour in the simulation (see Figure 7). Our simulation results highlight that even if the deadline is as small as one slot (10 minutes), we save around 40% energy consumption (over 24 hours) compared to without using dynamic deferral.

VII. EXTENSION FOR SOFT DEADLINES

The goal of this paper is to maximize energy saving from deferral of jobs. In the algorithms, deadline acts as a tool to ensure reasonable deferral times and to ensure eventual completion of each task before respective deadlines. It is important to recognize that a deadline may or may not be specified in great detail. The essence of this work is that workload deferral can be formulated effectively as a delayed task completion model. For this delay, our deadline determinism serves as a constraint for optimization purposes. This delay constraint can be specified either in terms of hard deadlines (as we have done in this paper) or by soft constraints e.g. average latency of job completion. In the case of soft deadline requirements, the deadline constraint (C1) in the formulation can be replaced by any latency constraint such as average latency which can be represented as: (C1) Latency Constraint: \[ \sum_{t=1}^{T} (L_j - x_j) \leq \xi, \] where \( 0 \leq \xi \leq 1 \), is a parameter to bound average latency. The remaining task is to experiment our algorithms with different latency constraints and compare the energy savings. We keep that as our future work.

VIII. RELATED WORK

With the importance of energy management in data centers, many scholars have applied energy-aware scheduling because of its low cost and practical applicability. In energy-aware scheduling, most work tries to find a balance between energy cost and performance loss through DVFS (Dynamic Voltage and Frequency Scaling) and DPM (Dynamic Power Management), which are the most common system-level power saving methods. Beloglazov et al. [26] gave the taxonomy and survey on energy management in data centers. Dynamic capacity provisioning is part of DPM technique. Most work on dynamic capacity provisioning for independent workload uses models based on queuing theory [27], [28], or control theory [29], [30]. Recently Lin et al. [6] used more general and common energy model and delay model and designed a provisioning algorithm for service jobs (e.g. HTTP requests) considering switching cost for the machines. They proposed a lazy capacity provisioning (LCP) algorithm which dynamically turns on/off servers in a data center to minimize energy cost and delay cost for scheduling workload. However their algorithm does not perform well for high peak-to-mean ratio (PMR) of the workload and does not provide bound on maximum delay. Moreover, LCP aims at minimizing the average delay while we regard latency as the deadline constraint. Instead of penalizing the delay, we purposely defer jobs within deadline in order to reduce the switching cost of the servers.

Many applications in real world require delay bound or deadline constraint e.g. see Lee et al. [25]. In the context of energy conservation, deadline is usually a critical adjusting tool between performance loss and energy consumption. Energy efficient deadline scheduling was first studied by Yao et al. [31]. They proposed algorithms, which aim to minimize energy consumption for independent jobs with deadline constraints.

In the context of data center, most prior work on energy management merely talks about minimizing the average delay without any bound on the delay. Recently, Mukherjee et al. [5]...
proposed online algorithms considering deadline constraints to minimize the computation, cooling and migration energy for machines. Goiri et al. [32] considered only batch jobs and proposed GreenSlot which predicts the amount of solar energy that will be available in near future and schedules the workload to maximize the green energy consumption while meeting the jobs’ deadlines. However these works are on job assignment problem and not on dynamic resource provisioning problem, where the number of needed servers is given in advance.

Recently researchers have proposed scheduling with deferral to improve performance of MapReduce jobs [10], [23]. Although MapReduce was designed for batch jobs, it has been increasingly used for small time-sensitive jobs. Delay scheduling with performance goals was proposed by Zaharia et al. [10] for scheduling jobs inside a Hadoop cluster with given resources. Verma et al. introduced a SLA-driven scheduling and resource provisioning framework considering given soft-deadline requirements for the MapReduce jobs [8], [9]. In a shared execution environment, Jockey [33] proposed utility-based resource allocation that ensures jobs are completed by importance. In relation to these works, we consider deadlines and schedule jobs within those deadlines and provision capacity to save energy. Recently, Chen et al. [12] identified a large class of interactive MapReduce workload and proposed policies for scheduling batch and small interactive jobs in separate clusters without any provisioning mechanism for the machines in those clusters. In contrast, we propose provisioning algorithms for the mix of batch and interactive jobs under bounded latency with constant competitive ratio.

IX. CONCLUSION

We have shown that significant reduction in energy consumption can be achieved by dynamic deferral of workload for capacity provisioning inside data centers. We have proposed two new algorithms, VFW(δ) and GCP, for provisioning the capacity and scheduling the workload while guaranteeing the deadlines. The algorithms use the flexibility in the latency requirements of the workload for energy savings and guarantee bounded cost and bounded latency under very general settings - arbitrary workload, general deadline and general energy cost models. Further both the VFW(δ) and GCP algorithms are simple to implement and do not require significant computational overhead. Additionally, the algorithms have constant competitive ratios and offer noteworthy cost savings as proved by theory, validated by simulation and demonstrated by experimentation. Although we have used MapReduce workload for validation, our algorithms can be applied for any workload as data centers have separate (physical/virtual) clusters for MapReduce and non-MapReduce jobs. The provisioning can be done on each such cluster. In order to save energy, the data center providers should provision their capacity (physical/virtual) by utilizing the flexibilities from SLAs via dynamic deferral.

REFERENCES

[1] Server and Data Center Energy Efficiency, Final Report to Congress, U.S. Environmental Protection Agency, 2007.

[2] Z. Liu, M. Lin, A. Wierman, S. Low, and L. Andrew, Greening Geographical Load Balancing, in Proc. ACM SIGMETRICS, 2011.

[3] C. Stewart and K. Shen, Some Joules Are More Precious Than Others: Managing Renewable Energy in the Datacenter, in Proc. Power Aware Comput. and Sys., October 2009.

[4] E. Pakbaznia and M. Pedram, Minimizing data center cooling and server power costs, in Proc. ISLPED, 2009.

[5] T. Mukherjee, A. Banerjee, G. Varsamopoulos, and S. K. S. Gupta, Spatio-Temporal Thermal-Aware Job Scheduling to Minimize Energy Consumption in Virtualized Heterogeneous Data Centers, Computer Networks, 53(17), 2009.

[6] L. Lin, A. Wierman, L. H. Andrew, E. Thereska, Dynamic right-sizing for power-proportional data centers, in Proc. IEEE INFOCOM, 2011.

[7] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, F. Zhao, Energy-aware server provisioning and load dispatching for connection-intensive internet services, in Proc. NSDI, 2009.

[8] A. Verma, L. Cherkasova, R. Campbell, SLO- Driven Right-Sizing and Resource Provisioning of MapReduce Jobs, in Proc. LADIS, 2011.

[9] A. Verma, L. Cherkasova, R. Campbell, Resource Provisioning Framework for MapReduce Jobs with Performance Guarantees, Middleware, 2011.

[10] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling, in Proc. EuroSys, 2010.

[11] Y. Chen, A. Ganapathi, R. Griffith and R. Katz, The Case for Evaluating MapReduce Performance Using Workload Suites, in Proc. IEEE MASS- COTS, 2011.

[12] Y. Chen, S. Alspaugh, D. Borthakur and R. Katz, Energy Efficiency for Large-Scale MapReduce Workloads with Significant Interactive Analysis, in Proc. EuroSys, 2012.

[13] D. Xu and X. Liu, Geographic trough filling for Internet datacenters, in Proc. of IEEE INFOCOM, 2012.

[14] J. Dean and S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, Comm. of the ACM, 51(1), pp. 107-113, 2008.

[15] M. A. Adnan, R. Sugihara, Y. Ma and R. Gupta, Dynamic Deferral of Workload for Capacity Provisioning in Data Centers, UCSD Technical Report, CoRR, abs/1109.3839, 2012.

[16] SPEC power data on SPEC website at http://www.spec.org.

[17] F. Bodik, M. P. Armbrust, K. Canini, A. Fox, M. Jordan, D. A. Patterson, A case for adaptive datacenters to conserve energy and improve reliability, UC Berkeley Tech. Report UCB/EECS-2008-127, 2008.

[18] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, Optimal power allocation in server farms, in Proc. of ACM SIGMETRICS, 2009.

[19] C. B. Lee, and A. Snavely, Precise and realistic utility functions for user-centric performance analysis of schedulers, in Proc. HFDC, 2007.

[20] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya, A taxonomy and survey of energy-efficient data centers and cloud computing systems, Advances in Computers, Elsevier: Amsterdam, 2011.

[21] A. Š. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, Optimal power allocation in server farms, in Proc. ACM Sigmetrics, 2009.

[22] M. A. Adnan, R. Sugihara, Y. Ma and R. Gupta, Dynamic Deferral of Workload for Capacity Provisioning in Data Centers, UCSD Technical Report, CoRR, abs/1109.3839, 2012.

[23] SPEC power data on SPEC website at http://www.spec.org.

[24] F. Bodik, M. P. Armbrust, K. Canini, A. Fox, M. Jordan, D. A. Patterson, A case for adaptive datacenters to conserve energy and improve reliability, UC Berkeley Tech. Report UCB/EECS-2008-127, 2008.

[25] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, Optimal power allocation in server farms, in Proc. of ACM SIGMETRICS, 2009.

[26] C. B. Lee, and A. Snavely, Precise and realistic utility functions for user-centric performance analysis of schedulers, in Proc. HFDC, 2007.

[27] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya, A taxonomy and survey of energy-efficient data centers and cloud computing systems, Advances in Computers, Elsevier: Amsterdam, 2011.

[28] A. Š. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, Optimal power allocation in server farms, in Proc. ACM Sigmetrics, 2009.