Simple and Effective Dynamic Provisioning for Power-Proportional Data Centers

Tan Lu and Minghua Chen
Department of Information Engineering, The Chinese University of Hong Kong

***Abstract*—Energy consumption represents a significant cost in data center operation. A large fraction of the energy, however, is used to power idle servers when the workload is low. Dynamic provisioning techniques aim at saving this portion of the energy, by turning off unnecessary servers. In this paper, we explore how much gain knowing future workload information can bring to dynamic provisioning. In particular, we develop online dynamic provisioning solutions with and without future workload information available. We first reveal an elegant structure of the off-line dynamic provisioning problem, which allows us to characterize the optimal solution in a "divide-and-conquer" manner. We then exploit this insight to design two online algorithms with competitive ratios $2-\alpha$ and $e/(e-1+\alpha)$, respectively, where $0 \le \alpha \le 1$ is the normalized size of a look-ahead window in which future workload information is available. A fundamental observation is that *future workload information beyond the full-size look-ahead window (corresponding to $\alpha = 1$) will not improve dynamic provisioning performance*. Our algorithms are decentralized and easy to implement. We demonstrate their effectiveness in simulations using real-world traces.**

I. Introduction

As Internet services, such as search and social networking, become more widespread in recent years, the energy consumption of data centers has been skyrocketing. In 2005, data centers worldwide consumed an estimated 152 billion kilowatt-hours (kWh) of energy [1]. Power consumption at such level was enough to power half of Italy [2]. Nowadays, the data center energy cost is growing 12% annually [3].

To reduce the power consumption of data centers, it is critical to save the energy consumed by servers [4], [5]. Real-world statistics suggests that ample saving is possible in server energy consumption [6], [7], [8], [4]. These statistics reveal that a large portion of the energy consumed by servers goes into powering nearly-idle servers, and it can be best saved by turning off servers during the off-peak periods.

One promising technique exploiting the above insights is *dynamic provisioning*, which turns on a minimum number of servers to meet the current demand and dispatches the load among the running servers to meet Service Level Agreements (SLA). The objective is to make the data center "power-proportional", i.e., use power only in proportion to the load.

There have been a substantial amount of efforts in developing such technique, initiated by the pioneering works [6], [7] a decade ago. Among them, one line of works [4], [8] examine the practical feasibility and advantage of dynamic provisioning using real-world traces. Another line of works [6], [9], [8] focus on developing algorithms with performance guarantee. See [10] for a recent survey.

The effectiveness of these exciting schemes, however, usually rely on being able to predict future workload to certain extent, e.g., using model fitting to forecast future workload from historical data [8]. This naturally leads to the following fundamental questions:

- Can we design solutions that require zero future workload information, called *online* solutions, yet still achieve *close-to-optimal* performance?
- Can we characterize the benefit of knowing future workload in dynamic provisioning?

Recently, Lin *et al.* [9] propose an algorithm that requires almost-zero future workload information and achieves a competitive ratio of 3, i.e., the energy consumption is at most 3 times the minimum (computed with perfect future knowledge). In simulations, they further show the algorithm can exploit available future workload information to improve the performance. These results are very encouraging, indicating that a complete answer to the questions is possible.

In this paper, we further explore answers to the questions, and make the following contributions:

- Under the setting that a running server consumes a fixed amount of energy per unit time[1], we reveal an elegant structure of the dynamic provisioning problem that allows us to solve it in a "divide-and-conquer" manner.
- We show that, interestingly, the optimal solution can be attained by the data center adopting a simple *last-empty-server-first* job-dispatching strategy and each server *independently* solving a classic ski-rental problem. We build upon this architectural insight to design two online algorithms. The first one, named **CSR**, is a deterministic algorithm with competitive ratio $2-\alpha$, where $0 \le \alpha \le 1$ is the normalized size of a look-ahead window in which future workload information is available. The second one, named **RCSR**, is a randomized algorithm with competitive ratio $e/(e-1+\alpha)$. Compared to the solution LCP(w) with competitive ratio 3, our algorithms have better competitive ratios and have guaranteed performance improvement when future information is available (corresponding to increasing α).
- Our results lead to a fundamental observation that *future workload information beyond the full-size look-ahead window (corresponding to $\alpha = 1$) will not improve dynamic provisioning performance*. The window size

[1] In [5], we extend our algorithms and results to the setting where we only assume that an idle server consumes a fixed amount of energy per unit time.

corresponding to $\alpha = 1$ is determined by the wear-and-tear cost and the unit-time cost of running one server.
- Our algorithms are simple and easy to implement. We demonstrate the effectiveness of our algorithms in simulations using real-world traces. We also compare their performance with state-of-the-art solutions.

Due to space limitation, all proofs are in [5].

II. Problem Formulation

A. Settings and Models

We consider a data center consisting of a set of homogeneous servers. Each server has a unit service capacity and can only serve one unit workload per unit time. Each server, when it is on, consumes P energy per unit time. We define β_{on} and β_{off} as the cost of turning a server on and off, respectively. Such wear-and-tear cost, including the amortized service interruption and hard-disk failure[11], is comparable to the energy cost of running a server for several hours [9].

We consider the following two types of workload:

- "mice" type of workload, such as "request-response" web serving. Each job of this type has a small transaction size and short duration. Many existing works [6], [7], [9] model such workload by a discrete-time fluid model. In the model, time is chopped into equal-length slots. Jobs arriving in one slot get served in the same slot. Workload can be split arbitrarily among running servers like fluid.
- "elephant" type of workload, such as virtual machine hosting in cloud computing. We model such workload by a continuous-time brick model. In this model, time is continuous. Jobs arrive and depart at arbitrary time, and no two job arrival/departure events happen simultaneously. We assume one server can only serve one job[2].

In the following, we present our results for the "elephant" type of workload. We show our algorithms and results are also applicable to "mice" type of workload [5].

Let $x(t)$ and $a(t)$ be the number of "on" servers (serving or idle) and jobs at time t, respectively. Under our workload model, $a(t)$ at most increases or decreases by one at any t.

Let $P_{on}(t_1, t_2)$ and $P_{off}(t_1, t_2)$ be the total cost of turning on and off servers in the time window $[t_1, t_2]$, respectively. It is clear that $P_{on}(t_1, t_2)$ and $P_{off}(t_1, t_2)$ are linearly proportional to the number of turning-on/-off times in $[t_1, t_2]$, respectively.

B. Problem Formulation

We formulate the **SCP** (server cost problem) of minimizing server the operation cost in a data center in $[0, T]$ as follows:

$$\textbf{SCP}: \quad \min \quad P\int_0^T x(t)\,dt + P_{on}(0,T) + P_{off}(0,T) \tag{1}$$

$$\text{s.t.} \quad x(t) \geq a(t), \forall t \in [0,T], \tag{2}$$

$$x(0) = a(0), x(T) = a(T), \tag{3}$$

$$\text{var} \quad x(t) \in \mathbb{Z}^+, t \in [0,T], \tag{4}$$

[2]Other than the obvious reason that the service capacity can only fit one job, there could also be SLA in cloud computing that requires the job does not share the physical server with other jobs due to security concerns.

where $\mathbb{Z}^+$ denotes the set of non-negative integers. The objective is to minimize the sum of server run-time cost and on-off cost. Constraints in (2) say the service capacity must satisfy the demand. Constraints in (3) are the boundary conditions.

There are infinite number of integer variables $x(t)$, $t \in [0, T]$, in the problem **SCP**, which make it challenging to solve. Moreover, in practice we have to solve the problem without knowing the workload $a(t)$, $t \in [0, T]$ ahead of time.

III. Optimal Solution and Offline Algorithm

We study the off-line version of the server cost minimization problem **SCP**, where the workload $a(t)$ in $[0, T]$ is given. We first identify an elegant structure of its optimal solution, which allows us to solve the problem in a "divide-and-conquer" manner. We then derive a simple and decentralized algorithm, upon which we build our online algorithms.

A. Critical Times and Critical Segments

We define a critical interval which we will use later as

$$\Delta \triangleq \frac{\beta_{on} + \beta_{off}}{P}. \tag{5}$$

Given $a(t)$ in $[0, T]$, we identify a set of critical times $\left\{T_i^c\right\}_i$ and construct the *critical segments* as follows.

Critical Segment Construction Procedure:

First, traversing $a(t)$, we identify all the jobs arrival/departure epochs in $[0, T]$. The first critical time is $T_1^c = 0$. T_1^c can be a job-arrival epoch or job-departure epoch, or no job departs/arrives the system at T_1^c. If no job departs or arrives at T_1^c, T_1^c is considered as a job-arrival epoch. Next we find T_{i+1}^c inductively, given that T_i^c is known.

- If T_i^c is a job-arrival epoch, e.g., the first critical time, then T_{i+1}^c is the first job-departure epoch after T_i^c. One example is the epoch T_2^c in Fig. 1.
- If T_i^c is a job-departure epoch, we first try to find the first arrival epoch τ after T_i^c so that $a(\tau) = a\left(T_i^c\right)$. If no such τ exists, we set T_{i+1}^c to be the next job departure epoch. One example is the T_3^c in Fig. 1. Otherwise, we check further:
  - if $a(t) = a\left(T_i^c\right) - 1, \forall t \in \left(T_i^c, \tau\right)$, then we set $T_{i+1}^c = \tau$. One example is the epoch T_4^c in Fig. 1.
  - if $a(t) < a\left(T_i^c\right) - 1, \exists t \in \left(T_i^c, \tau\right)$ and $\tau - T_i^c \leq \triangle$, then we set $T_{i+1}^c = \tau$.
  - if $a(t) < a\left(T_i^c\right) - 1, \exists t \in \left(T_i^c, \tau\right)$ and $\tau - T_i^c > \triangle$, then T_{i+1}^c is the first job-departure epoch after T_i^c.

Upon reaching time epoch T, we find all, say M, critical times. We define the critical segments as the period between two consecutive critical times, i.e., $\left[T_i^c, T_{i+1}^c\right]$, $1 \leq i \leq M-1$.

Examples of these four types of segments are shown in Fig. 1. We observe that workload expresses interesting properties in these critical segments.

Proposition 1. *The workload $a(t)$ in any critical segment $\left[T_i^c, T_{i+1}^c\right]$ must be one of the following four types*:

- *Type-I:* workload is non-decreasing in $\left[T_i^c, T_{i+1}^c\right]$.

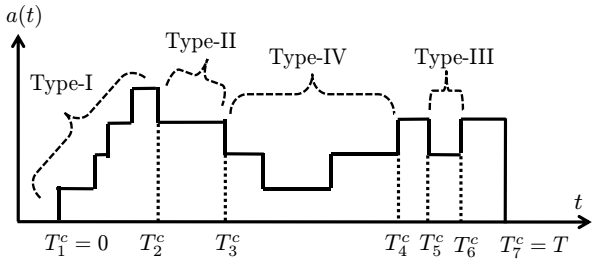


Figure 1: Illustration of critical times and critical segments. T_1^c to T_7^c are critical times, and they form six critical segments. $a(t)$ is of Type-I in $\left[T_1^c, T_2^c\right]$, Type-II in $\left[T_2^c, T_3^c\right]$, Type-III in $\left[T_5^c, T_6^c\right]$, and Type-IV in $\left[T_3^c, T_4^c\right]$.

- *Type-II:* workload is step-decreasing in $\left[T_i^c, T_{i+1}^c\right]$. That is, $a(t) = a\left(T_i^c\right) - 1, \forall t \in \left(T_i^c, T_{i+1}^c\right]$.
- *Type-III:* workload is of "U-shape" in $\left[T_i^c, T_{i+1}^c\right]$. That is, $a\left(T_{i+1}^c\right) = a\left(T_i^c\right)$ and $a(t) = a\left(T_i^c\right) - 1, \forall t \in \left(T_i^c, T_{i+1}^c\right)$.
- *Type-IV:* workload is of "canyon-shape" in $\left[T_i^c, T_{i+1}^c\right]$. That is, $a\left(T_{i+1}^c\right) = a\left(T_i^c\right)$, $a(t) \le a\left(T_i^c\right) - 1$ and not always identical, $\forall t \in \left(T_i^c, T_{i+1}^c\right)$. Moreover, we have $T_{i+1}^c - T_i^c \le \triangle$.

B. Structure of Optimal Solution

Let $x^*(t)$, $t \in [0, T]$, be an optimal solution to the problem **SCP**, and the corresponding minimum server operation cost be P^*. We have the following observation.

Lemma 2. *$x^*(t)$ must meet $a(t)$ at every critical time, i.e., $x^*\left(T_i^c\right) = a\left(T_i^c\right)$, $1 \le i \le M$.*

Lemma 2 not only presents a necessary condition for a solution $x(t)$ to be optimal, but also suggests a "divide-and-conquer" way to solve the problem **SCP** optimally.

Consider a sub-problem of minimizing the server operation cost in a critical segment $\left[T_i^c, T_{i+1}^c\right]$ with boundary condition $x\left(T_i^c\right) = a\left(T_i^c\right), x\left(T_{i+1}^c\right) = a\left(T_{i+1}^c\right)$, $1 \le i \le M-1$. This sub-problem is similar to the problem **SCP,** except that the time period is $\left[T_i^c, T_{i+1}^c\right]$ other than $[0, T]$. Let its optimal value be P_i^*, $1 \le i \le M-1$. We have the following observation.

Lemma 3. *$\sum_{i=1}^{M} P_i^*$ is a lower bound of the optimal server operation cost of the problem **SCP**, i.e.,*

$$P^* \ge \sum_{i=1}^{M} P_i^*. \tag{6}$$

Eqn. (6) establishes a lower bound of P^*, and as we will see soon, it is achievable. Suggested by Lemma 3, it suffices to solve individual sub-problems for all critical segments in $[0, T]$, and combine the corresponding solutions to form an optimal solution to the overall problem **SCP**.

Optimal Solution Construction Procedure:

We visit all the critical segments in $[0, T]$ sequentially, and construct an $x(t)$, $t \in [0, T]$. For a critical segment $\left[T_i^c, T_{i+1}^c\right]$, $1 \le i \le M-1$, we check the $a(t)$ in it:

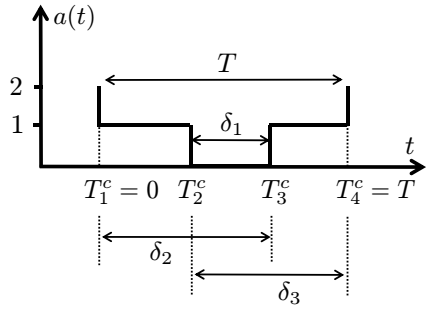


Figure 2: An example of a critical segment $[0, T]$. We define intervals $\delta_1 = T_3^c - T_2^c$, $\delta_2 = T_3^c - T_1^c$, and $\delta_3 = T_4^c - T_2^c$.

1) the $a(t)$ is of Type-I or Type-II: we simply set $x(t) = a(t)$, for all $t \in \left[T_i^c, T_{i+1}^c\right]$.
2) the $a(t)$ is of Type-III:
   - if $\Delta \ge T_{i+1}^c - T_i^c$, then we set $x(t) = a\left(T_i^c\right), \forall t \in \left[T_i^c, T_{i+1}^c\right]$;
   - otherwise, we set $x(T_i^c) = a(T_i^c)$, $x(T_{i+1}^c) = a(T_{i+1}^c)$, and $x(t) = a\left(T_i^c\right) - 1, \forall t \in \left(T_i^c, T_{i+1}^c\right)$.
3) the $a(t)$ is of Type-IV: we set $x(t) = a\left(T_i^c\right), \forall t \in \left[T_i^c, T_{i+1}^c\right]$.

The following theorem shows that the lower bound of P^* in (6) is achieved by using the above procedure.

Theorem 4. *The **Optimal Solution Construction Procedure** terminates in finite time, and the resulting $x(t)$, $t \in [0, T]$, is an optimal solution to the problem **SCP**.*

C. A Case Study and Insights

In the following, we go through the construction of $x(t)$ for a workload shown in Fig. 2, to bring out the intuition.

During the critical segment $[0, T]$ with the workload shown in Fig. 2, the system starts and ends with 2 jobs and 2 running servers. Let the servers with their jobs leaving at time 0 and T_3^c be S1 and S2, respectively.

At time 0, a job leaves. The procedure compares Δ and T. If $\Delta > T$, then it sets $x(t) = 2$ and keeps all two servers running for all $t \in [0, T]$; otherwise, it decomposes the interval $[0, T]$ into three small ones $[T_1^c, T_2^c]$, $[T_2^c, T_3^c]$, and $[T_3^c, T_4^c]$, as shown in Fig. 2. The first small critical segment $[T_1^c, T_2^c]$ has a Type-II workload, thus the procedure sets $x(t) = 1$ for $t \in [T_1^c, T_2^c]$. The second small segment $[T_2^c, T_3^c]$ has a Type-III workload; thus for all $t \in [T_2^c, T_3^c]$, the procedure maintains $x(t) = 1$ if $\Delta > \delta_1$ and sets $x(t) = 0$ otherwise. The last small segment $[T_3^c, T_4^c]$ has a Type-I workload, thus the procedure set $x(t) = 1$ for $t \in [T_3^c, T_4^c)$ and $x(T_4^c) = 2$.

These actions reveal two important observations, upon which we build our off-line algorithm.

- Newly arrived jobs should be assigned to servers in the reverse order of their last-empty-epochs.
- Upon being assigned an empty period, a server only needs to *independently* make locally energy-optimal decision.

The intuition behind the first observation is that job-dispatching should try to make every server empty as long

as possible so that the server-on-off option, if explored, can save abundant energy. The second observation allows us to decouple the decisions of individual servers.

D. Offline Algorithm Achieving the Optimal Solution

Exploiting the two observations made in the previous case-study subsection, we design a off-line algorithm that gives an optimal $x^*(t)$. While existing centralized off-line algorithms (e.g., the one in [9]) for solving an alternative formulation can also be modified to solve the problem **SCP**, our algorithm is unique in that it utilizes a new structure of the problem **SCP** to allow simple and decentralized implementation.

Decentralized Off-line Algorithm:

By a central job-dispatching entity: it implements a last-empty-server-first strategy. In particular, it maintains a stack (i.e., a Last-In/First-Out queue) storing the IDs for all idle or off servers. Before time 0, the stack contains IDs for all the servers that are not serving.

- Upon a job arrival: the entity pops a server ID from the top of the stack, and assigns the job to the corresponding server (if the server is off, the entity turns it on).
- Upon a job departure: a server just turns idle, the entity pushes the server ID into the stack.

By each server:

- Upon receiving a job: it serves the job immediately.
- Upon a job leaving this server and it becomes empty: let the current time be t_1. The server searches for the earliest time $t_2 \in (t_1, t_1 + \Delta]$ so that $a(t_2) = a(t_1)$. If no such t_2 exists, the server turns itself off. Otherwise, it stays idle.

The following theorem justifies the optimality of the off-line algorithm. The proof can be found in [5].

Theorem 5. *The proposed off-line algorithm achieves the optimal server operation cost of the problem **SCP**.*

There are two important observations. First, the job-dispatching strategy only depends on the past job arrivals and departures. Consequently, the strategy assigns a job to the same server no matter it knows future job arrival/departure or not; it also acts independently to servers' off-or-idle decisions. Second, each individual server is actually solving a classic ski-rental problem [12] – whether to "rent", i.e., keep idle, or to "buy", i.e., turn off now and on later, but with their "days-of-skiing" (corresponding to servers' empty periods) *jointly determined by the job-dispatching strategy*.

Next, we extend our proposed off-line algorithm to its online versions with performance guarantee.

IV. Online Dynamic Provisioning with and without Future Workload Information

We construct two online algorithms by combining (i) the same last-empty-server-first job-dispatching strategy as the one in our proposed off-line algorithm, and (ii) an off-or-idle decision module running on each server to *solve online ski-rental problems*. The following lemma presents an important observation on the last-empty-server-first job-dispatching strategy.

Lemma 6. *For the same $a(t), t \in [0, T]$, under the last-empty-server-first job-dispatching strategy, each server will get the same job at the same time and the job will leave the server at the same time for both off-line and online situations.*

As a result, *in the online case, each server still faces the same set of off-or-idle problems* as compared to the off-line case. This is the key to derive the competitive ratios of our online algorithms later on. Each server, not knowing the empty periods ahead of time, however, needs to decide whether to stay idle or be off (and if so when) in an online fashion. One natural approach is to adopt classic algorithms for the online ski-rental problem.

A. Dynamic Provisioning without Future Information

For the online ski-rental problem, the break-even algorithm in [12] and the randomized algorithm in [13] have competitive ratios 2 and $e/(e-1)$, respectively. The ratios have been proved to be optimal for deterministic and randomized algorithms, respectively. Directly adopting these algorithms in the off-or-idle decision module leads to two online solutions for the problem **SCP** with competitive ratios 2 and $e/(e-1) \approx 1.58$. The resulting solutions are decentralized and easy to implement: a central entity runs the last-empty-server-first job-dispatching strategy, and each server independently runs an online ski-rental algorithm. The competitive ratios of the two resulting solutions improve the best known ratio 3 achieved by the algorithm in [9].

B. Dynamic Provisioning with Future Information

In the data center dynamic provisioning problem, one key observation many existing solutions exploited is that the workload expressed highly regular patterns. Thus the workload information in a small look-ahead window can be accurately estimated by machine learning or model fitting based on historical data [8]. Can we exploit such future knowledge, if available, in designing online algorithms? If so, how much gain can we get? To answer these questions, we propose new future-aware online solutions with performance guarantee.

We model the availability of future workload information as follows. For any t, the workload $a(\tau)$ for in the look-ahead window $[t, t + \alpha\Delta]$ is known, where $\alpha \in [0, 1]$ is a constant and $\alpha\Delta$ represents the size of the look-ahead window. Due to the last-empty-server-first job-dispatching strategy, the server knows that it will receive a job in the window $[t, t + \alpha\Delta]$ if there exists a $\tau \in [t, t + \alpha\Delta]$ so that $a(\tau) > a(t)$, and no job otherwise. We first present a deterministic online algorithm named ***CSR*** *(Collective Server-Rentals)*.

Future-Aware Online Algorithm CSR:

By a central job-dispatching entity: it implements the last-empty-server-first job-dispatching strategy, i.e., the one described in the off-line algorithm.

By each server:

- Upon receiving a job: it serves the job immediately.

- Upon a job leaving this server and it becomes empty: the server waits for $(1-\alpha)\Delta$ amount of time,
  - if it receives a job during the period, it serves the job immediately;
  - otherwise, it looks into the look-ahead window of size $\alpha\Delta$. It turns itself off, if it will receive no job during the window. Otherwise, it stays idle.

Next, we present a randomized online algorithm named ***RCSR*** *(Randomized Collective Server-Rentals)* as follow.

Future-Aware Online Algorithm RCSR:
By a central job-dispatching entity: it implements the last-empty-server-first job-dispatching strategy.
By each server:

- Upon receiving a job: it serves the job immediately.
- Upon a job leaving this server and it turns empty: the server waits for Z amount of time, where Z is generated according to the following probability distribution

$$\begin{cases} f_Z(z) = \begin{cases} \frac{1-\frac{\alpha}{e-1+\alpha}}{(e-1)\Delta(1-\alpha)} e^{z/(1-\alpha)\Delta}, & \text{if } 0 < z \le (1-\alpha)\Delta; \\ 0, & \text{otherwise.} \end{cases} \\ P(Z=0) = \frac{\alpha}{e-1+\alpha} \end{cases}$$

  - if it receives a job during the period, it servers the job immediately;
  - otherwise, it looks into the look-ahead window of size $\alpha\Delta$. It turns itself off, if it will receive no job during the window. Otherwise, it stays idle.

The algorithms are decentralized, making them easy to implement and scale. Observing no such future-aware online algorithms available in the literature, we analyze their competitive ratios and present the results as follows.

Theorem 7. *For any positive P, β_{off}, and β_{on}, the online algorithms CSR and RCSR have competitive ratios of $2-\alpha$ and $e/(e-1+\alpha)$.*

Remark: Theorem 7 reveals a fundamental observation that future workload information beyond the full-size look-ahead window (corresponding to $\alpha = 1$) will not improve dynamic provisioning performance. This is because there exist algorithms, e.g., CSR and RCSR, that can make optimal decision to minimize power consumption when the future workload in the full-size look-ahead window is available.

V. Experiments

A. Settings

Workload trace: The real-world traces we use in experiments are a set of I/O traces taken from 6 RAID volumes at MSR Cambridge [14]. The traced period was one week from February 22 to 29, 2007. The jobs are "request-response" type and we use a discrete-time fluid model to describe the workload, with the slot length being 10 minutes and the load being the average number of jobs in each slot.

In the technical report [5], we show that our proposed algorithms also work with the discrete-time fluid workload model after simple modification; hence, we apply the modified algorithms to the above real-world traces.

Cost benchmark: Current data centers usually do not use dynamic provisioning. The cost incurred by static provisioning is usually considered as benchmark to evaluate new algorithms [9], [4]. Static provisioning runs a constant number of servers to serve the workload. In our experiment, we assume that the data center has the complete workload information ahead of time and provisions just to satisfy the peak load. Using such benchmark gives us an estimate of the maximum cost saving.

Sever operation cost: The server operation cost is determined by unit-time energy cost P and on-off costs β_{on} and β_{off}. In the experiment, we assume that a server consumes one unit energy for per unit time, i.e., $P = 1$. Similar to [9], we set $\beta_{off} + \beta_{on} = 6$. Under this setting, the critical interval is $\Delta = \left(\beta_{off} + \beta_{on}\right)/P = 6$ units of time.

B. Performance of CSR and RCSR

We have characterized in Theorem 7 the competitive ratios of CSR and RCSR as the look-ahead window size, i.e., $\alpha\Delta$, increases. The resulting competitive ratios, i.e., $2-\alpha$ and $e/(e-1+\alpha)$, although quite appealing, are for the worst-case scenarios. In practice, the actual performance can be even better than what the ratios suggest.

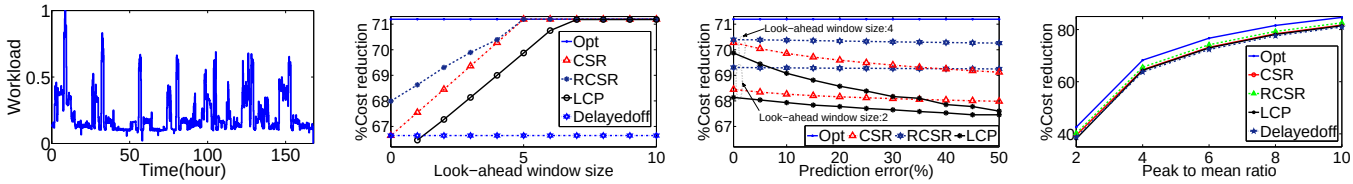
In our first experiment, we study the performance of CSR and RCSR. The results are shown in Fig. 3b. The vertical axis indicates the cost reduction and the horizontal axis indicates the size of look-ahead window. The cost reduction curves are obtained by comparing the power cost incurred by the offline algorithm, CSR, RCSR, the LCP(w) algorithm [9] and the DELAYEDOFF algorithm [15] to the cost benchmark.

As seen, for this set of workload, all four online algorithms achieve substantial cost reduction as compared to the benchmark. In particular, CSR and RCSR achieve 66% saving even without future workload information. LCP(w) has to have (or estimate) one unit time of future workload to execute, and thus it starts to perform when the look-ahead window size is one. The cost reductions of CSR and RCSR grow as the look-ahead window increases, and reaching optimal when the look-ahead window size reaches Δ. These observations match what Theorem 7 predicts. Meanwhile, LCP(w) has not yet reached the optimal performance. DELAYEDOFF has the same performance for all look-ahead window sizes since it does not exploit future workload information.

C. Impact of Prediction Error

Previous experiments show that CSR, RCSR and LCP(w) have better performance if accurate future workload is available. However, there are always prediction errors in practice. Therefore, it is important to evaluate the performance of the algorithms in the present of prediction error through experiment. Theoretical analysis on the impact of prediction error is an interesting topic for future investigation.

In particular, we evaluate CSR and RCSR with look-ahead window size of 2 and 4 units of time. Zero-mean Gaussian prediction error is added to each unit-time workload in the look-ahead window, with its standard deviation grows from



(a) MSR data trace for one week (b) Impact of future information (c) Impact of prediction error (d) Impact of PMR

Figure 3: Real-world workload trace and the performance of algorithms under different settings. The critical interval Δ is 6 units of time. We discuss the performance of algorithms CSR, RCSR, LCP(w) and DELAYEDOFF in Section V-E.

0 to 50% of the corresponding actual workload. In practice, prediction error tends to be small [16]; thus we are stress-testing the algorithms. We average 100 runs for each algorithm and show the results in Fig. 3c.

On one hand, we observe all algorithms are fairly robust to prediction errors. On the other hand, all algorithms achieve better performance with look-ahead window size 4 than size 2. This indicates more future workload information, even inaccurate, is still useful in boosting the performance.

D. Impact of Peak-to-Mean Ratio (PMR)

Intuitively, dynamic provisioning can save more power when the data center trace has large PMR. Our experiments confirm this intuition which is also observed in other works [9], [4]. Similar to [9], we generate the workload from the MSR traces by scaling $a(t)$ as $\overline{a(t)} = Ka^{\gamma}(t)$, and adjusting γ and K to keep the mean constant. We run the off-line algorithm, CSR, RCSR, LCP(w) and DELAYEDOFF using workloads with different PMRs ranging from 2 to 10, with look-ahead window size of one unit time. The results are shown in Fig. 3d. As seen, energy saving increases from about 40% at PRM=2, which is common in large data centers, to large values for the higher PMRs that is common in small to medium sized data centers. Similar results are observed for different look-ahead window sizes.

E. Comparison with LCP(w) and DELAYEDOFF

Compared to LCP(w) with competitive ratio 3 regardless of whether future information is available, CSR and RCSR have better competitive ratios ($2-\alpha$ and $e/(e-1+\alpha)$) and have improved performance when future information is available (corresponding to α increasing to 1). Although better ratios may not assure that CSR and RCSR have better performance for all types of workload since the competitive ratio is a worst-case metric, Fig. 3b shows that CSR and RCSR perform better than LCP(w) for the data trace from MSR Cambridge. The performance gains of CSR and RCSR over LCP(w) and DELAYEDOFF shown in Fig. 3b may seem small. However, these gains, when multiplying the huge amount of energy consumed by the data centers worldwide every year, correspond to a substantial amount of energy cost saving beyond the state-of-the-art solutions. Moreover, since our algorithms' competitive ratios are much smaller than that of LCP(w), there exist cases that CSR and RCSR perform substantially better than LCP(w).

Compared to DELAYEDOFF, CSR and RCSR have performance guarantee in terms of competitive ratios and they improve as future information is made available. In contrast, DELAYEDOFF does not have a competitive ratio analysis and does not explore future information to improve performance.

Acknowledgments

We thank Minghong Lin and Lachlan Andrew for sharing the code of their LCP algorithm, and Eno Thereska for sharing the MSR Cambridge data center traces. This work is supported by a China 973 Program (Project No. 2012CB315904), and Hong Kong General Research Fund grants (Project No. 411209, 411010, and 411011), and an Hong Kong Area of Excellence Grant (Project No. AoE/E-02/08).

References

[1] J. G. Koomey, "Worldwide electricity used in data centers," *Environmental Research Letters*, no. 3, 2008.
[2] I. E. Agency, "World energy balances (2007 edition)," 2007.
[3] U.S. Environmental Protection Agency, "Epa report on server and data center energy efficiency," *ENERGY STAR Program*, 2007.
[4] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, and R. Katz, "Napsac: design and implementation of a power-proportional web cluster," *ACM SIGCOMM Computer Communication Review*, 2011.
[5] T. Lu and M. Chen, "Simple and effective dynamic provisioning for power-proportioinal data centers," CUHK, Tech. Rep., 2011. [Online]. Available: http://arxiv.org/pdf/1112.0442v2.pdf
[6] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," in *Proc. ACM SOSP*, 2001.
[7] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath, "Load balancing and unbalancing for power and performance in cluster-based systems," in *Workshop on Compilers and Operating Systems for Low Power*, 2001.
[8] 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 *Proc. USENIX NSDI*, 2008.
[9] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *Proc. IEEE INFOCOM*, 2011.
[10] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Univ. of Melbourne, Tech. Rep. CLOUDS-TR-2010-3*, 2010.
[11] H. Qian and D. Medhi, "Server operational cost optimization for cloud computing service providers over a time horizon," in *USENIX Hot'ICE*, 2011.
[12] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator, "Competitive snoopy caching," *Algorithmica*, 1988.
[13] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki, "Competitive randomized algorithms for nonuniform problems," *Algorithmica*, vol. 11, no. 6, pp. 542–571, 1994.
[14] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage (TOS)*, vol. 4, no. 3, p. 10, 2008.
[15] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Performance Evaluation*, 2010.
[16] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, 2009.