Simple and Effective Dynamic Provisioning for Power-Proportional Data Centers

Tan Lu, Minghua Chen, and Lachlan L. H. Andrew

***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

Cloud computing is a new paradigm for providing Internet services to a large volume of end-users. In this paradigm, cloud computing service providers provide infrastructure, in particular data centers, as a service and charge customers based on their usage. However, the energy consumption of data centers hosting these services has been skyrocketing. In 2010, data centers worldwide consumed an estimated 240 billion kilowatt-hours (kWh) of energy, roughly 1.3% of the world total energy consumption [2]. Power consumption at such a level is almost enough to power all of Spain [3]. Energy-related costs are approaching the cost of IT hardware in data centers [4], and are growing 12% annually [5].

Recent work has explored electricity price fluctuation in time and geographically balancing load across cloud data centers to cut the electricity costs; see e.g., [6], [7], [8], [9] and the references therein. To benefit from this, the energy consumption of a data center must reflect its actual load.

Energy consumption in a data center is a product of the power usage effectiveness (PUE)[1] and the energy consumed by the servers. There have been substantial efforts in improving PUE, e.g., by optimizing cooling [10], [11] and power management [12]. In this paper, we focus on reducing the energy consumed by the servers.

A preliminary version of the paper appeared in CISS in 2012[1]. Tan Lu and Minghua Chen are with Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong. Lachlan L. H. Andrew is with the Centre for Advanced Internet Architectures, Swinburne University of Technology, Australia.

[1]PUE is defined as the ratio between the amount of power entering a data center and the power used to run its computer infrastructure. The closer to one PUE is, the better energy utilization is.

Real-world statistics reveals three observations that suggest ample saving is possible in server energy consumption [13], [14], [15], [16], [17], [18]. First, workload in a data center often fluctuates significantly on the timescale of hours or days, expressing a large "peak-to-mean" ratio. Second, data centers today often provision for far more than the observed peak to accommodate both the predictable workload and the unpredictable flash crowds. Such static over-provisioning results in low average utilization for most servers. Third, a lightly-utilized or idle server consumes more than 60% of its peak power. These observations imply 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. In particular, an important technique for reducing the energy consumption of idle servers is for servers to autonomously turn off sub-systems [19].

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), making the data center "power-proportional". This is enabled by virtualization, which is the fundamental technology that allows the cloud to exist.

There has been a significant amount of effort in developing such technique, initiated by the pioneering works [13], [14] a decade ago. Among them, one line of work [19], [16], [15] examines the practical feasibility and advantage of dynamic provisioning using real-world traces, suggesting substantial gain is indeed possible in practice. Another line of work [13], [20], [15] focuses on developing algorithms by utilizing various tools from queuing theory, control theory and machine learning to provide insights that can lead to effective solutions. These existing works provide a number of schemes that deliver favorable performance justified by theoretic analysis and/or practical evaluations. See [21] for a recent survey.

However, turning servers on and off incurs a cost. Hence the effectiveness of these exciting schemes usually relies on the ability to predict future workload to a certain extent, e.g., using model fitting to forecast future workload from historical data [15]. This naturally leads to the following questions:

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

Answers to these questions provide fundamental understanding on how much performance gain one can have by exploiting future workload information in dynamic provisioning.

The worst-case performance of an on-line algorithm A is often measured by its competitive ratio: the maximum, over all possible problem instances, of ratio of the cost of the solution found by A to the cost of the optimal off-line solution. Recently, Lin *et al.* [22] proposed an algorithm that requires almost-zero future workload information[2] 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:

- We consider a scenario where a running server consumes a fixed amount of energy per unit time. We reveal that the dynamic provisioning problem has an elegant structure that allows us to solve it in a "divide-and-conquer" manner. This insight leads to a full characterization of the optimal solution, achieved by a centralized procedure.
- We show that the optimal solution can also be attained by 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 *decentralized* online algorithms. The first, named CSR, is deterministic with competitive ratio $2-\alpha$, where $0 \leq \alpha \leq 1$ is the normalized size of a look-ahead window in which future workload information is available. The second, named RCSR, is randomized with competitive ratio $e/(e-1+\alpha)$. We prove that $2-\alpha$ and $e/(e-1+\alpha)$ are the best competitive ratios for deterministic and randomized online algorithms under *last-empty-server-first* job-dispatching strategy.
- Our results lead to a fundamental observation: under the cost model that a running server consumes a fixed amount of energy per unit time, *future workload information beyond the full-size look-ahead window will not improve the dynamic provisioning performance.* The size of the full-size look-ahead window is determined by the wear-and-tear cost and the unit-time energy cost of one server.
- We also extend the algorithms to the case where servers take setup time T_s to turn on and workload $a(t)$ satisfies $a(\tau) \leq (1+\gamma)a(t)$ for all $\tau \in [t, t+T_s]$, achieving competitive ratios upper bounded by $(2-\alpha)(1+\gamma)+2\gamma$ and $\frac{e}{e-1+\alpha}(1+\gamma)+2\gamma$.
- 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.

The rest of the paper is organized as follows. We formulate the problem in Section II. Section III reveals the important structure of the formulated problem, characterizes the optimal solution, and designs a simple decentralized offline algorithm achieving the optimal. In Section IV, we propose online algorithms and provide performance guarantees. Section V presents the experiments and Section VI concludes the paper.

[2]LCP algorithm [22] is a discrete time algorithm that only requires an estimate of the job arrival rate of the current slot.

II. Problem Formulation

A. Settings and Models

We consider a data center consisting of a set of homogeneous servers. Without loss of generality, we assume each server has a unit service capacity[3], i.e., it can only serve one unit workload per unit time. Let the unit time power consumption of busy and idle servers be P_b and P, respectively. We define β_{on} and β_{off} as the cost of turning a server on and off, respectively. This includes wear-and-tear costs, including the amortized service interruption cost and procurement and replacement cost of server components (hard-disks and power supplies in particular). [20], [23]. It is comparable to the energy cost of running a server for several hours [22].

The results we develop in this paper apply to both of the following two types of workload:

- "mice" type workloads, such as "request-response" web serving. Each job of this type has a small transaction size and short duration. A number of existing works [13], [14], [22], [24] model such workloads by a discrete-time fluid model. In the model, time is divided into equal-length slots. Jobs arriving in one slot get served in the same slot. Workload can be split among running servers at arbitrary granularity like a fluid.
- "elephant" type workloads , such as virtual machine hosting in cloud computing. Each job of this type has a large transaction size, and can last for a long time. We model such workload by a continuous-time brick model. In this model, time is continuous, and we assume one server can only serve one job[4]. Jobs arrive and depart at arbitrary times, and no two job arrival/departure events happen simultaneously.

For the discrete-time fluid model, servers toggled at the discrete time epoch will not interrupt job execution and thus no job migration is incurred. This neat abstraction allows research to focus on server on-off scheduling to minimize the cost. For the continuous-time brick model, when a server is turned off, the long-lasting job running on it needs to be migrated to another server. In general, such non-trivial migration cost needs to be taken into account when toggling servers.

In the following, we present our results based on the continuous-time brick model. We add discussions to show the algorithms are also applicable to the discrete-time fluid model.

We assume that each job is present on a closed interval of time. The number of jobs as a function of time is then a non-negative, integer valued upper semi-continuous function a. For convenience, we further assume that a changes by at most 1

[3]In practice, server's service capacity can be determined from the knee of its throughput and response-time curve [16].

[4]This could be justified if there were a SLA in cloud computing that requires the job does not share the physical server with other jobs due to security concerns. The problem is substantially different if a single server can host multiple virtual machines (VMs). Specifically, if the scheduling discipline is restricted to being non-clairvoyant (job sizes are only known when they complete) then VM migration becomes much more beneficial than in the case that scheduling discipline is clairvoyant; without VM migration, the competitive ratio is at least as large as the number of VMs that can be hosted on a single server in the case that scheduling discipline is non-clairvoyant.

at any time. To avoid technicalities, we assume a is bounded and not always zero.

The number of servers "on" (serving or idle) can be defined as follows. For each server s, define a function u_s that is right continuous with $u_s(0^-) = 0$, counting the number of times the server has turn on, and a function d_s that is left continuous with and $d_s(0) = 0$, counting the number of times the server has turned off. The state of server s at time t is then $x_s(t) = u_s(t) - d_s(t)$, which must be either 0 or 1. Then define $u = \sum_s u_s$and $d = \sum_s d_s$. The total number of servers on is $x = u - d$.

To focus on the cost within $[0, T]$, we require $x(0) = a(0)$ and $x(T) = a(T)$. For convenience, we set $a(t) = 0$ for all $t < 0$ and all $t > T$.

Define the cost of server s on an interval $[t_1, t_2)$ as

$$c_s(t_1, t_2) = P \int_{t_1}^{t_2} x_s(t)\, dt + \beta_{on}(u_s(t_2) - u_s(t_1)) + \beta_{off}(d_s(t_2) - d_s(t_1)) \tag{1}$$

where the integral represents the running cost, and the other terms are the switching costs. Note that this includes any cost of switching on at t_2 even though that is not in the interval, and neglects the cost of switching off at $t_1$even though that is in the interval. Consequently, for any $t_1 < t_2 < t_3$ we have $c_s(t_1, t_3) = c_s(t_1, t_2) + c_s(t_2, t_3)$.

It will sometimes be useful to consider the entire switching cost on a closed interval. Let $P_{on}(t_1, t_2)$ and $P_{off}(t_1, t_2)$ denote the total wear-and-tear cost incurred by turning on and off servers in $[t_1, t_2]$, respectively. They take the on-off cost at t_1 and t_2 into account. Specifically, if u has left limits and d has right limits, then $P_{on}(t_1, t_2) = \beta_{on}(u_s(t_2) - u_s(t_1^-))$ and $P_{off}(t_1, t_2) = \beta_{off}(d_s(t_2^+) - d_s(t_1))$. Our results depend only on the sum $P_{on} + P_{off}$, but we retain both terms to emphasize the two physical processes.

B. Problem Formulation

We formulate the problem of minimizing server operation cost in a data center in an interval $[T_1, T_2]$ given an initial number of "on" servers $X_1$and a final number of "on" servers X_2 as follows:

$$\begin{aligned}
& \mathcal{P}[a(t), X_1, X_2, T_1, T_2]: \\
\min \quad & P \int_{T_1}^{T_2} x(t)\, dt + P_{on}(T_1, T_2) + P_{off}(T_1, T_2) && (2)\\
\text{s.t.} \quad & x(t) \geq a(t), \forall t \in [T_1, T_2], && (3)\\
& x(T_1) = X_1, x(T_2) = X_2, && (4)\\
\text{var} \quad & x(t) \in \mathbb{Z}^+, t \in [T_1, T_2], && (5)
\end{aligned}$$

where $\mathbb{Z}^+$ denotes the set of non-negative integers. In particular, we are interested in the "Server Capacity Provisioning" problem, SCP, given by $\mathcal{P}[a(t), a(0), a(T), 0, T]$.

The objective is to minimize the sum of server energy consumption and the wear-and-tear cost. The actual summation of the two parts of the cost is $\int_0^T P[x(t) - a(t)] + P_b a(t)\, dt + P_{on}(0, T) + P_{off}(0, T)$, since the busy and idle powers can differ. However $\int_0^T P_b a(t) - P a(t)\, dt$ is constant for given $a(t)$, and so to minimize the total cost is to minimize (2). The constraints in (3) say the service capacity must satisfy the demand. The constraints in (4) are the boundary conditions.

Remarks:

1) The problem SCP does not consider the possible migration cost associated with the continuous-time discrete-load model. Fortunately, our results later show that we can schedule servers according to the optimal solution, and at the same time dispatch jobs to servers in a way that aligns with their on-off schedules, thus incurring no migration cost. Hence, the minimum server operation cost remains unaltered even we consider migration cost in the problem SCP (which can be rather complicated to model).
2) The formulation remains the same with the discrete-time fluid workload model where there is no job migration cost to consider.
3) The problem SCP is similar to a common one considered in the literature, e.g., in [22], with a specific (linear) cost function. The benefit of SCP is that we retain the constraint that the decision variables be integers instead of real numbers. This is important for clusters and small data centers.

There are an infinite number of integer variables $x(t)$, $t \in [0, T]$, in the problem SCP, which make it challenging to solve. Moreover, in practice the data center has to solve the problem without knowing the workload $a(t)$, $t \in [0, T]$ ahead of time. In reality, $a(t)$ is not continuous and it may be right continuous or left continuous at all the discontinuous points. However, in our SCP problem, we will modify $a(t)$ to make it right continuous when $a(t)$ increases by one left continuous when $a(t)$ decreases by one. This simple modification will not change the optimal value of SCP.

Next, we design an off-line algorithm, including (i) a job-dispatching algorithm and (ii) a server on-off scheduling algorithm, to solve the problem SCP optimally. We then extend the algorithm to its on-line versions and analyze their performance guarantees with or without (partial) future workload information.

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 design a procedure to construct an optimal solution to problem SCP. We then derive a simple and decentralized algorithm, upon which we build our online algorithms.

A. Structure of Optimal Solution

We first define the "critical interval" as

$$\Delta \triangleq \frac{\beta_{on} + \beta_{off}}{P} \tag{6}$$

Let M be the maximum value of $a(t)$, $t \in [0, T]$. We then define $\bar{a}(t)$,$t \in [-2\Delta, T + 2\Delta]$ as an extension of $a(t)$:

$$\bar{a}(t) = \begin{cases} a(t) & t \in [0, T] \\ 0 & t \in (-2\Delta, 0) \cup (T, T + 2\Delta) \\ M + 1 & t \in \{-2\Delta, T + 2\Delta\} \end{cases}$$

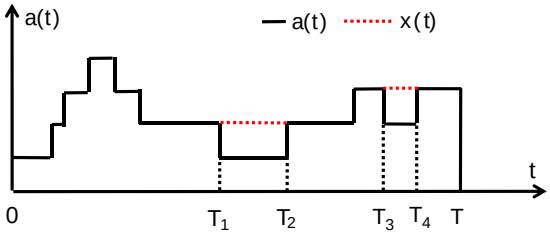


Figure 1: Example of solution constructed by Optimal Solution Construction Procedure.

Let $x^*(t)$, $t \in [0, T]$, be an optimal solution to the problem SCP, and the corresponding minimum server operation cost be P^*. The optimal solution can be constructed as follows.

Optimal Solution Construction Procedure:

For A from 1 to $M+1$ **do**

Find all the intervals (τ, τ')in $[-2\Delta, T+2\Delta]$ such that $\bar{a}(\tau) \geq A$, $\bar{a}(\tau') \geq A$ and $\bar{a}(t) < A, \forall t \in (\tau, \tau')$.

For all intervals (τ, τ') **do**

If $\tau' - \tau \leq \Delta$ **then**

(re)assign $x(t) \leftarrow \min[\bar{a}(\tau), \bar{a}(\tau')]$;

Else

for any part of the interval that $x(t)$ has not already been set, set $x(t) \leftarrow \bar{a}(t)$.

End if

End For

End For

One example of $x(t), t \in [0, T]$ can be found in Fig. 1.

The following theorem is proved in Appendix A using proof-by-contradiction and counting arguments.

Theorem 1. *The result of the Optimal Solution Construction Procedure, $x(t)$, $t \in [0, T]$, is an optimal solution to the problem SCP. Moreover, the optimal u_s and d_s have both left and right limits.*

B. Intuitions and Observations

Consider the example shown in Fig. 2. During $[0, T]$, the system starts and ends with two jobs and two running servers. Let the servers whose jobs leave at times 0 and T be S1 and S2, respectively.

At time 0, a job leaves. Let T be the time T until $a(t)$ again reaches the level $a(0)$. The procedure compares T against Δ. If $\Delta > T$, then it sets $x(t) = 2$ and keeps all two servers running for all $t \in [0, T]$; otherwise, according to Optimal Solution Construction Procedure, $x(t) = 1$ for $t \in [0, T_1] \cup [T_2, T]$ and $x(t) = 0, \forall t \in [T_1, T_2]$ if $\Delta > \delta_1$ or $x(t) = 0, \forall t \in [T_1, T_2]$ if $\Delta \leq \delta_1$.

These actions reveal two important observations, upon which we build a decentralized off-line algorithm to solve the problem SCP optimally.

- 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

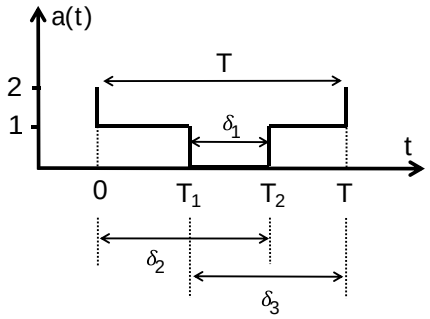


Figure 2: An example of a time period $[0, T]$. Interval $\delta_1 = T_2 - T_1$, $\delta_2 = T_2$, and $\delta_3 = T - T_1$.

In the example, when a new job arrives at time T_2, the procedure implicitly assigns it to server S2 instead of S1. As a result, S1 and S2 have empty periods of T and δ_1, respectively. This may sound "unfair" compared to an alternative strategy that assigns the job to the early-emptied server S1, which gives S1 and S2 empty periods of δ_2 and δ_3, respectively. However, at each decision point, allocating to the last-empty server results in a distribution of the idle times I that is "convexly larger" than that resulting from any other allocation; i.e., it maximizes $E[(I-x)^+]$ for all x [25]. Note that if x is the time after which the server decides to sleep, then $E[(I-x)^+]$ is the expected energy saving.

It is straightforward to verify that in the example, upon a job leaving server S1 at time 0, the procedure implicitly assigns an empty-period of T to S1, and turns S1 off if $\Delta < T$ and keeps it running at idle state otherwise. Similarly, upon a job leaving S2 at time T_1, S2 is turned off if $\Delta < \delta_1$ and stays idle otherwise. Such comparisons and decisions can be done by individual servers themselves.

C. Offline Algorithm Achieving the Optimal Solution

The Optimal Solution Construction Procedure determines how many running servers to maintain at time t, i.e., $x^*(t)$, to achieve the optimal server operation cost P^*. However, as discussed in Section II-A, under the continuous-time brick model, scheduling servers on/off according to $x^*(t)$ might incur non-trivial job migration cost.

Exploiting the two observations made in the case-study at the end of last subsection, we design a simple and decentralized off-line algorithm that gives an optimal $x^*(t)$ and *incurs no job migration cost*.

Decentralized Off-line Algorithm:

A central job-dispatching entity 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 identifiers (IDs) for all the servers that are not serving.

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

- Upon a job departure: a server becomes idle, and the dispatcher pushes the server ID into the stack.

Each server:

- Upon receiving a job: the server starts serving the job immediately.
- Upon a job leaving this server: let the departure epoch be t_1. The server searches for the earliest time $t_2 \in (t_1, t_1+\Delta]$ such that $a(t_2) = a(t_1)$. If no such t_2 exists, then the server turns itself off. Otherwise, it stays idle.

We remark that in the algorithm, we use the same server to serve a job during its entire sojourn time. Thus there is no job migration cost. The following theorem justifies the optimality of the off-line algorithm.

Theorem 2. *The proposed off-line algorithm achieves the optimal cost of the problem SCP.*

Proof: Refer to Appendix B. ■

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 whether it knows future job arrival/departure times or not; it also acts independently from servers' off-or-idle decisions. Second, each individual server is actually solving a classic "ski-rental" (rent-or-buy) problem [26]. Each chooses whether to "rent"(keep idle and pay an ongoing energy cost) or to "buy" (turn off now and pay a on-off wear-and-tear cost), but the number of rounds is *jointly determined by the job-dispatching strategy*.

Next, we exploit these two observations to extend the off-line algorithm to its online versions with performance guarantees.

IV. Online Dynamic Provisioning with or without Future Workload Information

Inspired by our off-line algorithm, we construct online algorithms by combining (i) the same last-empty-server-first job-dispatching strategy as the one in the proposed off-line algorithm, and (ii) an off-or-idle decision module running on each server to *solve an online ski-rental problem*. To evaluate our online algorithms, we compare its performance to that of the best off-line algorithm. We say a deterministic online algorithm A is R-competitive if for all input sequences σ, we have

$$C_A(\sigma) \leq RC_{opt}(\sigma) + O(1)$$

where $C_A(\sigma)$ is the cost of algorithm A and $C_{opt}(\sigma)$ is the offline optimal. We say a randomized online algorithm A, is R-competitive [5] if for all input sequences σ, we have

$$\mathbb{E}[C_A(\sigma)] \leq RC_{opt}(\sigma) + O(1)$$

where $\mathbb{E}[C_A(\sigma)]$ is the expectation of the cost of algorithm A with respect to its random choices for input sequence σ, and $C_{opt}(\sigma)$ is the offline optimal.

[5]against an "oblivious adversary"

As discussed at the end of last section, the last-empty-server-first job-dispatching strategy utilizes only past job arrival/departure information. Consequently in both the offline and online cases, it assigns the same set of jobs to the same server at the same sequence of epochs.

Lemma 3. *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.*

Lemma 3 is true because last-empty-server-first job-dispatching strategy only depends on past workload and it is independent of the past statuses of servers.

As a result, *in the online case, each server still faces the same set of off-or-idle problems* as in the off-line case. This is the key to deriving the competitive ratios of the online algorithms we will present.

Each server, not knowing the empty periods ahead of time, needs to decide whether to stay idle or turn 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 Workload Information

For the online ski-rental problem, the break-even algorithm in [26] and the randomized algorithm in [27] 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 $e/(e-1) \approx 1.58$. These ratios improve the best known ratio 3 achieved by the algorithm in [22].

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. For example, if the break-even algorithm is used, a server that just becomes empty at time t will stay idle for an amount of time Δ. If it receives no job during this period, it turns itself off. Otherwise, it starts to serve the job immediately. As a special case covered by Theorem 5, it turns out this directly gives a 2-competitive dynamic provisioning solution.

B. Dynamic Provisioning with Future Workload Information

Studies of online problems usually assume zero future information. However, in our data center dynamic provisioning problem, one key observation many existing solutions exploited is that the workload exhibits highly regular patterns. Thus the workload information in a near look-ahead window may be accurately estimated by machine learning or model fitting based on historical data [15], [28]. Can we exploit such future knowledge, if available, in designing online algorithms? If so, how much gain can we get?

Let us elaborate through an example to explain why and how much future knowledge can help. Suppose at any time t, the workload information $a(t)$ in a look-ahead window $[t, t+\alpha\Delta]$

is available, where $\alpha \in [0,1]$ is a constant. Consider a server running the break-even algorithm that becomes empty at time t_1, and has an empty period infinitesimally longer than Δ.

Following the standard break-even algorithm, the server waits for time Δ before turning itself off. In this example, it receives a job immediately after $t_1 + \Delta$ epoch, and it has to power up to serve the job. This incurs a total cost of $2P\Delta$ as compared to the optimal one $P\Delta$, which is achieved by the server staying idle all the way.

Another strategy that costs less is as follows. The server stays idle for an amount of time $(1-\alpha)\Delta$, and peeks into the look-ahead window $[t_1 + (1-\alpha)\Delta, t_1 + \Delta]$. Due to the last-empty-server-first job-dispatching strategy, the server can easy tell that it will receive a job if any $a(t)$ in the window exceeds $a(t_1)$, and no job otherwise. In this example, the server sees itself receiving no job during $[t_1 + (1-\alpha)\Delta, t_1 + \Delta]$ and it turns itself off at time $t_1 + (1-\alpha)\Delta$. Later it turns itself on to serve the job right after $t_1 + \Delta$. Under this strategy, the overall cost is $(2-\alpha)P\Delta$, which is better than that of the break-even algorithm.

This example shows it is possible to modify classic online algorithms to exploit future workload information to obtain better performance. To this end, we propose new future-aware online ski-rental algorithms and build new online solutions.

We model the availability of future workload information as follows. For any t, the workload in the window $[t, t + \alpha\Delta]$ is known, where $\alpha \in [0,1]$ is a constant and $\alpha\Delta$ represents the size of the window.

We present both the modified break-even algorithm and the decentralized and *deterministic* online solution named *CSR (Collective Server-Rentals)* as follow. The modified future-aware break-even algorithm is very simple and is summarized as the part in the server's actions upon job departure.

Future-Aware Online Algorithm CSR:
A central job-dispatching entity implements the last-empty-server-first job-dispatching strategy, i.e., the one described in the off-line algorithm.
Each server:

- Upon receiving a job, the server starts serving the job immediately.
- Upon a job leaving this server and it becomes empty, the server waits an amount of time $(1-\alpha)\Delta$.
  - If it receives a job during the period, it starts serving 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.

In fact, as shown in Theorem 5 later in this section, the algorithm CSR has the best possible competitive ratio for any deterministic algorithms. Thus, no deterministic algorithms can achieve better competitive ratio than the algorithm CSR.

The competitive ratio can be improved by replacing the deterministic sleep decision by a randomized decision, similarly to [27], but extended to consider future information. However, if servers are turned off after random times, then it is possible that the last-empty server is off even though there are idle servers. Instead of using the last-empty server, we will dispatch the job to the server, if any, that is on but has been idle least time, as done in [29].

The following decentralized and *randomized* online algorithm named *RCSR (Randomized Collective Server-Rentals)* is new, and has the best possible competitive ratio.

Future-Aware Online Algorithm RCSR:
A central job-dispatching entity implements the least-idle job-dispatching strategy.
Each server:

- Upon receiving a job: resets all timers, and start serving the job immediately.
- Upon a job leaving this server: records the occupancy as a, and initializes a timer to expire time Z into the future, where Z has probability density

$$f_Z(z) = \frac{\exp(z/\{(1-\alpha)\Delta\})}{(e-1+\alpha)(1-\alpha)\Delta}\mathbf{1}_{0<z\le(1-\alpha)\Delta} + \frac{\alpha}{e-1+\alpha}\delta(z) \tag{7}$$

  where δ is the Dirac delta distribution, and $\mathbf{1}_X = 1$ if X is true, and 0 otherwise.
- Upon expiration of the timer, consult the prediction engine. If the maximum occupancy in the coming window of size $\alpha\Delta$ is less than a, then turn off. Otherwise, remain idle until a job is assigned.

The following lemma, proved in Appendix C, shows that RCSR performs at least as well as it would under the last-empty-server-first job-dispatching strategy, which will allow us to obtain a competitive ratio.

Lemma 4. *For any given workload, the cost of using RCSR is, with probability 1, no greater than the cost of applying last-empty-server-first with the same per-server sleep policy, provided that the same random number Z is generated under both schemes for any given job departure.*

The two future-aware online algorithms inherit the nice properties of the proposed off-line algorithm in the previous section. The same server is used to serve a job during its entire sojourn time. Thus there is no job migration cost. Although the job-dispatching entity in our algorithms is centralized, it just maintains a stack and estimates future workload. The majority of the algorithm is performed by the (decentralized) servers. This makes CSR and RCSR easy to implement and scalable.

Observing no such future-aware online algorithms available in the literature, we analyze their competitive ratios and present the results as follows. Assume that jobs assigned to a server is countable.

Theorem 5. *For any $P, \beta_{off}, \beta_{on}$, the online algorithms CSR and RCSR have competitive ratio of $2-\alpha$ and $e/(e-1+\alpha)$. $2-\alpha$ and $e/(e-1+\alpha)$ are the best competitive ratios for deterministic and randomized algorithms for SCP, under last-empty-server-first job-dispatching strategy, respectively.*

Proof: Refer to Appendix D. ■

Remarks: (i) When $\alpha = 1$, both two algorithms achieve the optimal server operation cost. This matches the intuition

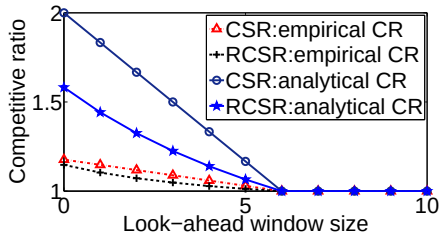


Figure 3: Comparison of the worst-case competitive ratios (according to Theorem 5) and the empirical competitive ratios observed in simulations using real-world traces. The full-size look-ahead window size $\Delta = 6$ units of time. More simulation details are in Section V.

that servers only need to look Δ amount of time ahead to make optimal off-or-idle decision upon job departures. This immediately gives a fundamental insight that future workload information beyond the critical interval Δ (corresponding to $\alpha = 1$) will not improve dynamic provisioning performance. (ii) The competitive ratios presented in the above theorem are for the worst case. We have carried out simulations using real-world traces and found the empirical ratios are much better, as shown in Fig. 3. (iii) To achieve better competitive ratios, the theorem says that it is necessary to change the job-dispatching strategy, since otherwise no deterministic or randomized algorithms do better than the algorithms CSR and RCSR. (iv) Our analysis assumes the workload information in the look-ahead window is accurate. We evaluate the two online algorithms in simulations using real-world traces with prediction errors, and observe they are fairly robust to the errors. More details are provided in Section V.

Note that our algorithms are closely related to the DELAYEDOFF algorithm in [29], despite the fact that they seek to optimize different objective functions (total energy consumption in our study v.s. Energy-Response time Product (ERP) in [29]). The main algorithmic difference is that we make use of future information to improve performance, and use randomization to improve the competitive ratio. The main analytic difference is that we consider worst-case performance, whereas [29] considers expected performance in a stochastic setting and a large-system asymptotic regime.

C. Adapting the Algorithms to Work with Discrete-Time Fluid Workload Model

Adapting our off-line and online algorithms to work with the discrete-time fluid workload model involves two simple modifications. Recall in the discrete-time fluid model, time is divided into equal-length slots. Jobs arriving in one slot get served in the same slot. Workload can be split among running servers at arbitrary granularity like a fluid.

For the job-dispatching entity in all the algorithms, at the end of each slot when all servers are considered to be empty, it pushes all the server IDs back into the stack (order doesn't matter). Then at the beginning of each slot, it pops just-enough server IDs from the stack in a Last-In/First-Out manner to satisfy the current workload. In this way, the job-dispatching entity essentially packs the workload to as few servers as possible, following the last-empty-server-first strategy.

Each individual server starts to serve upon receiving a job, and starts to solve the (off-line or online) discrete ski-rental problem upon the job leaving. It is not difficult to verify the modified algorithms still retain their corresponding performance guarantees. Specifically:

Corollary 6. *The modified deterministic and randomized online algorithms for discrete-time fluid workload have competitive ratios of* $2-\alpha$ *and* $e_\theta/(e_\theta - 1 + \alpha)$*, respectively, where* θ *is the difference between the length of break-even interval and look-ahead window and* $e_\theta = (1 + 1/\theta)^\theta \uparrow e$ *as* $\theta \to \infty$.

D. Extending to Case Where Servers Have Setup Time.

Until now, we have ignored the time T_s required for a server to turn on. The competitive ratio will be unbounded unless there is a bound on the number of jobs that can arrive during T_s. We now describe a centralized algorithm EXT that provides a bounded CR in the case where $a(\tau) \le (1+\gamma)a(t)$ for all $\tau \in [t, t+T_s]$. We expect γ to be small [30]. In this model, servers can be in three states: ON, BOOT, OFF. Only servers in state ON can serve jobs, but servers in states ON and BOOT both consume power P per unit time. An OFF server "turns on" when it enters state BOOT; T_s later it will become ON. A server in any state can immediately be turned OFF.

Algorithm EXT for Cases with Setup Time:
Each server:

- Behaves as for CSR or RCSR, but when its timer expires, it does not turn off but sends a message M to manager.

Manager:

Keeps track of the set X (of size x) of "active" servers, i.e., those that have not sent M since being allocated a job. It responds to two types of events as follows:

- Job arrival: If X contains an idle server, the job is sent to a server in X using least-idle. Otherwise it is sent to another ON server. Additional servers will be turned on so that the total number of ON and BOOT servers is $\lfloor x(1+\gamma) \rfloor + 1$.
- Message M from server: All but $\lceil x(1+\gamma) \rceil + 1$ servers will be turned OFF. BOOT servers are turned off first, in decreasing order of how recently they were turned on. No active servers are turned off.

The following result, proven in Appendix E, establishes the validity and performance guarantees of EXT.

Corollary 7. *If there are* $\lceil a(0)(1+\gamma) \rceil + 1$ *ON servers at time 0, then under EXT, the number of ON servers at time* t *is at least* $a(t)$ *. Let* $a_{\min} = \min_{t\in[0,T]} a(t)$*. The competitive ratio of EXT on instances with discrete arrival instants is* $(2-\alpha)(1+\gamma) + 2/a_{min}$ *if servers use CSR, or* $\frac{e}{e-1+\alpha}(1+\gamma) + 2/a_{min}$ *if servers use RCSR. These are bounded above by* $(2-\alpha)(1+\gamma) + 2\gamma$ *and* $\frac{e}{e-1+\alpha}(1+\gamma) + 2\gamma$.

Remarks: (i) The competitive ratio of EXT is linearly proportional to γ. (ii) Since the minimal workload a_{min} in

large data centers is normally much larger than that in small ones, whence EXT is usually more beneficial for large data center. (iii) In EXT, we adopt over-provisioning to combat the problem that servers need setup time T_s, however, future workload may not be known; it would be interesting to know if there exist other approaches to handle this problem. EXT can not achieve competitive ratio of 1 even if $\alpha = 1$. Therefore, it is also good to know how to better utilize future information.

V. Experiments

We implement the proposed off-line and online algorithms and carry out simulations using real-world traces to evaluate their performance. Our aims are threefold. First, to evaluate the performance of the algorithms in a typical setting. Second, to study the impacts of workload prediction error and workload characteristics on the algorithms' performance. Third, to compare our algorithms to two recently proposed solutions LCP(w) in [22] and DELAYEDOFF in [29].

A. Settings

Workload trace: The traces we use in experiments are a set of I/O traces taken from 6 RAID volumes at MSR Cambridge [31]. The traced period was one week from February 22 to 29, 2007. We estimate the average number of jobs over disjoint 10 minute intervals. The data trace has a peak-to-mean ratio (PMR) of 4.63. The jobs are "request-response" type and thus the workload is better described by a discrete-time fluid model, with the slot length being 10 minutes and the load in each slot being the average number of jobs.

In the experiments, we run algorithm LCP(w) [22] by directly using the above discrete-time trace, since LCP(w) was originally designed to work under a discrete-time setting. Meanwhile, CSR, RCSR, and DELAYEDOFF [29] were primally designed to work under a continuous-time setting. To evaluate their performance by using the above discrete-time trace, we run these algorithms by feeding jobs continuously to the algorithms, where the job-arrivals in a slot are assumed to uniformly spread out the slot. By this setting, we would like to demonstrate that algorithms CSR/RCSR/DELAYEDOFF do not require to know the number of job-arrivals a priori to operate. We use last-empty-server-first for RCSR.

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 [22], [16]. Static provisioning runs a constant number of servers to serve the workload. In order to satisfy the time-varying demand during a period, data centers usually overly provision and keep more running servers than what is needed to satisfy the peak load. In our experiment, we assume that the data center has the complete workload information ahead of time and provisions exactly to satisfy the peak load. Using such benchmark gives us a conservative estimate of the cost saving from our algorithms.

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., $P = 1, \forall x$. We set $\beta_{off} + \beta_{on} = 6$, i.e., the cost of turning a server off and on once is equal to that of running it for six units of time [22]. Under this setting, the critical interval is $\Delta = \left(\beta_{off} + \beta_{on}\right)/P = 6$ units of time.

B. Performance of the Proposed Online Algorithms

We have characterized in Theorem 5 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 $e/(e - 1 + \alpha)$, already appealing, are for the worst case. In practice, the actual performance can be even better.

In our first experiment, we study the performance of CSR and RCSR using real-world traces. The results are shown in Fig. 4b. The cost reduction curves are obtained by comparing the cost incurred by the off-line algorithm, CSR, RCSR, the LCP(w) algorithm [22] and the DELAYEDOFF algorithm [29] to the cost benchmark. The vertical axis indicates the cost reduction and the horizontal axis indicates the size of look-ahead window varying from 0 to 10 units of time.

For this workload, CSR, RCSR, LCP(w) and DELAYEDOFF achieve substantial cost reduction as compared to the benchmark. In particular, the cost reductions of CSR and RCSR are beyond 66% even when no future workload information is available. LCP(w) starts to perform when the look-ahead window size is one. This is because we run LCP(w) under a discrete-time setting and the workload information for the current slot is only available after all jobs in this slot have arrived. Meanwhile, CSR, RCSR, and DELAYEDOFF are running under a continuous-time setting, where jobs arriving at any moment are served immediately.

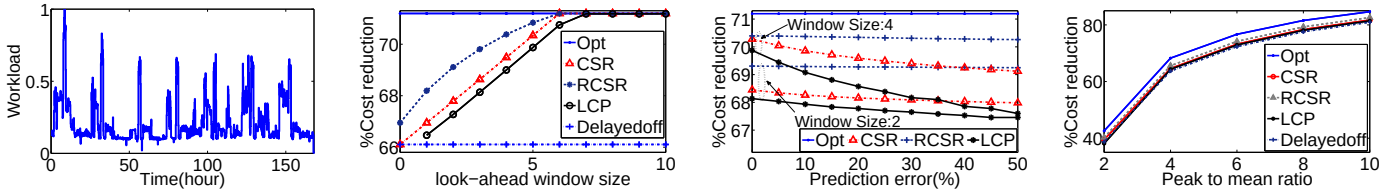
The cost reductions of CSR and RCSR grow linearly as the look-ahead window increases, and reaching optimal when the look-ahead window size reaches Δ. These observations match what Theorem 5 predicts. Meanwhile, LCP(w) has not yet reach the optimal performance when the look-ahead window size reaches the critical value Δ. 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.

To achieve this goal, 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 0 to 50% of the corresponding actual workload. In practice, prediction error tends to be small [32]; thus we are essentially stress-testing the algorithms.

We average 100 runs for each algorithm and show the results in Fig. 4c, where the vertical axis represents the cost reduction as compared to the benchmark.



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

Figure 4: 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.

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, comparing to static provisioning, 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 [22], [16]. Similar to [22], 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. 4d.

As seen, energy saving increases form 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. Discussion

Note that CSR and RCSR have competitive ratios $2-\alpha$ and $e/(e-1+\alpha)$, which improve as future information is available. This is in contrast to LCP(w), whose best known competitive ratio is 3 and, regardless of how much future information is available, there are instances with performance arbitrarily close to the ratio. Fig. 4b shows that CSR/RCSR perform slightly better than LCP(w), partially because they need not work in discrete time. These performance gains of CSR/RCSR over LCP(w) and DELAYEDOFF shown in Fig. 4b, when multiplying the large amount of energy consumed by the data centers every year, correspond to non-negligible energy cost saving. Moreover, the sleep management in CSR/RCSR are decentralized, which makes them very much easier to implement; while the LCP(w) is inherently centralized, since it requires the solution of a convex program at each time.

Although in this example, DELAYEDOFF performs close to the optimal, there are very natural cases in which it can be almost a factor of two more expensive than CSR/RCSR. The value of Δ is approximately one hour [22], and it is common for workloads to have a periodic structure with period one hour. In this case, it is possible that DELAYEDOFF always turns machines off just before they are needed again. If the workload can be predicted an hour into the future, then CSR/RCSR can guarantee optimal performance in this case. DELAYEDOFF also does not exploit randomness to improve performance like RCSR does.

VI. Concluding Remarks

Dynamic provisioning is an effective technique for reducing server energy consumption in data centers, by turning off unnecessary servers to save energy. In this paper, we design online dynamic provisioning algorithms with zero or partial future workload information available.

We reveal an elegant "divide-and-conquer" structure of the off-line dynamic provisioning problem, under the cost model that a running server consumes a fixed amount of energy per unit time. Exploiting such structure, we show its optimal solution can be achieved 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 new decentralized online algorithms. One is deterministic with competitive ratio $2-\alpha$, where $0 \le \alpha \le 1$ is the fraction of the full-size look-ahead window in which future workload information is available. The size of the full-size look-ahead window is determined by the wear-and-tear cost and the unit-time energy cost of running a single server. The other is randomized with competitive ratio $e/(e-1+\alpha)$. The ratios $2-\alpha$ and $e/(e-1+\alpha)$ are the best competitive ratios for any deterministic and randomized online algorithms under last-empty-server-first job-dispatching strategy. Note that the problem we study in this paper is similar to that studied in [22]. The difference is that we optimize a linear cost function over integer variables, while Lin et al. in [22] minimize a convex cost function over continuous variables (by relaxing the integer constraints). This paper and [22] obtain different online algorithms with different competitive ratios for the two different formulations, respectively.

Our results lead to a fundamental observation that under the cost model that a running server consumes a fixed amount of energy per unit time, future workload information beyond the the full-size look-ahead window will not improve the dynamic provisioning performance.

In addition, we also propose online algorithms for the case that servers need setup time T_s but the load satisfies $a(\tau) \le (1+\gamma)a(t)$ for all $\tau \in [t, t+T_s]$. These algorithms have competitive ratios $(2-\alpha)(1+\gamma)+2\gamma$ and $\frac{e}{e-1+\alpha}(1+\gamma)+2\gamma$.

Our algorithms are simple and easy to implement. Simulations using real-world traces show that our algorithms can achieve close-to-optimal energy-saving performance, and are robust to future-workload prediction errors.

These results suggest that it is possible to reduce server energy consumption significantly with zero or only partial future workload information.

This work can be extended in many important directions. In the elephant model considered here, each server could only serve one job at a time. Cloud data centres typically run multiple VMs on each physical machine. One particular motivation is to pack together jobs with complementary resource requirements, such as placing a CPU-intensive and a memory-intensive VM on the same server. In this scenario, minimizing the total power cost is a dynamic bin-packing problem which is NP-hard. (It contains classic bin packing as a special case.). The analysis of dynamic bin-packing problem is entirely different and it would be interesting to look at it in the future. Even in the simplest case that each server can host an arbitrary combination of m VMs, the problem is significantly different; it is no longer the case that the optimal performance can be obtained by a non-clairvoyant algorithm without VM migration, and indeed such algorithms are at best m- competitive. A related extension would be to consider the fact that VMs may have time-varying resource requirements.

Another important direction would be to extend these results to the case of heterogeneous servers or multiple geographically separated data centers [33], [34], [35]. It would be useful to extend the insight from this paper to heterogeneous cases.

Acknowledgements

We thank Minghong Lin for sharing the code of his LCP algorithm, and Eno Thereska for sharing the MSR Cambridge data center traces. Tan Lu and Minghua Chen's research is partially supported by a the China 973 Program (Project No. 2012CB315904), the General Research Fund (Project No. 411209, 411010, and 411011) and an Area of Excellence Grant (Project No. AoE/E-02/08), all established under the University Grant Committee of the Hong Kong SAR, China, as well as an Open Project of Shenzhen Key Lab of Cloud Computing Technology and Application and two gift grants from Microsoft and Cisco. This work was also funded by ARC grant FT0991594.

References

[1] T. Lu and M. Chen, "Simple and effective dynamic provisioning for power-proportional data centers," in *Proceedings of the 46th Annual Conference on Information Sciences and Systems (CISS)*, 2012.

[2] J. G. Koomey, "Growth in data center electricity use 2005 to 2010," *Oakland, CA: Analytics Press*, 2010.

[3] Spain energy consumption. [Online]. Available: http://www.nationmaster.com/country/sp-spain/ene-energy

[4] L. Barroso, "The price of performance," *ACM Queue*, vol. 3, no. 7, pp. 48–53, 2005.

[5] U.S. Environmental Protection Agency, "Epa report on server and data center energy efficiency," *ENERGY STAR Program*, 2007.

[6] Z. Liu, M. Lin, A. Wierman, S. Low, and L. Andrew, "Greening geographical load balancing," in *Proc. ACM SIGMETRICS*, 2011, pp. 233–244.

[7] P. Wendell, J. Jiang, M. Freedman, and J. Rexford, "Donar: decentralized server selection for cloud services," in *Proc. ACM SIGCOMM*, vol. 40, no. 4, 2010, pp. 231–242.

[8] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," in *Proc. ACM SIGCOMM*, 2009, pp. 123–134.

[9] R. Urgaonkar, B. Urgaonkar, M. Neely, and A. Sivasubramaniam, "Optimal power cost management using stored energy in data centers," in *Proc. ACM SIGMETRICS*, 2011, pp. 221–232.

[10] N. Rasmussen, "Electrical efficiency modeling of data centers," *Technical Report White Paper*, vol. 113.

[11] R. Sharma, C. Bash, C. Patel, R. Friedrich, and J. Chase, "Balance of power: Dynamic thermal management for internet data centers," *IEEE Internet Computing*, 2005.

[12] 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*, vol. 36, no. 1, 2008, pp. 48–59.

[13] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," in *Proc. ACM SOSP*, 2001.

[14] 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.

[15] 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.

[16] 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*, vol. 41, no. 1, pp. 102–108, 2011.

[17] X. Fan, W. Weber, and L. Barroso, "Power provisioning for a warehouse-sized computer," in *Proc. the 34th annual international symposium on Computer architecture*, 2007.

[18] L. Barroso and U. Holzle, "The case for energy-proportional computing," *IEEE Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[19] D. Meisner, B. Gold, and T. Wenisch, "Powernap: eliminating server idle power," *ACM SIGPLAN Notices*, 2009.

[20] H. Qian and D. Medhi, "Server operational cost optimization for cloud computing service providers over a time horizon," in *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, 2011, pp. 4–4.

[21] Y. C. L. A. Beloglazov, R. Buyya and A. Zomaya, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in Computers, M. Zelkowitz(ed.), vol. 82, pp. 47-111*, 2011.

[22] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *Proc. IEEE INFOCOM, Shanghai, China*, pp. 10–15, 2011.

[23] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *ACM SIGMETRICS*, vol. 33, no. 1, 2005, pp. 303–314.

[24] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat, "Model-based resource provisioning in a web service utility," in *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, 2003.

[25] I. Meilijson and A. Nádas, "Convex majorization with an application to the length of critical paths," *Journal of Applied Probability*, pp. 671–677, 1979.

[26] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator, "Competitive snoopy caching," *Algorithmica*, vol. 3, no. 1, pp. 79–119, 1988.

[27] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki, "Competitive randomized algorithms for nonuniform problems," *Algorithmica*, vol. 11, no. 6, pp. 542–571, 1994.

[28] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters," in *Proceedings of the 2009 conference on Hot topics in cloud computing*.

[29] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Performance Evaluation*, 2010.

[30] V. Mathew, R. Sitaraman, and P. Shenoy, "Energy-aware load balancing in content delivery networks," *Proc. IEEE INFOCOM*, 2012.

[31] 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.

[32] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[33] M. Lin, Z. Liu, A. Wierman, and L. L. H. Andrew, "Online algorithms for geographical load balancing," in *Proc. Int. Green Computing Conf.*, 2012.
[34] R. Nathuji, C. Isci, and E. Gorbatov, "Exploiting platform heterogeneity for power efficient data centers," in *Autonomic Computing, 2007. ICAC'07. Fourth International Conference on*. IEEE, 2007, pp. 5–5.
[35] T. Heath, B. Diniz, E. Carrera, W. Meira Jr, and R. Bianchini, "Energy conservation in heterogeneous server clusters," in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2005, pp. 186–195.
[36] C. Mathieu, "Online algorithms:ski rental." [Online]. Available: http://www.cs.brown.edu/~claire/Talks/skirental.pdf

Appendix

The claims made in the previous sections will now be proven from A to E.

A. Proof of Theorem 1

In order to prove theorem 1, we introduce three lemmas. The first establishes that P_{on} and P_{off} are well defined.

Lemma 8. *The optimal u_s and d_s have both left and right limits.*

Proof: The interval between two discontinuities in the optimal d_s (or optimal u_s) is at least Δ, and so the set of discontinuities has no accumulation points. Since it is piecewise constant, this is sufficient for it to have both left and right limits at all points. ■

Lemma 9. *Let $m = \max\{\bar{a}(\tau) : \tau \in (T_s, T_e)\}$. If $T_s - T_e > \Delta$ and $m < \min(\bar{a}(T_s), \bar{a}(T_e))$ then a necessary condition for $x(t)$ to achieve optimal cost of $\mathcal{P}(\bar{a}, X, Y, T_s, T_e)$ is that $x(t) \leq m, \forall t \in (T_s, T_e)$.*

Proof: Let $x_i(t)$ be any optimal solution to the above optimization problem $\mathcal{P}(\bar{a}, X, Y, T_s, T_e)$ and $x_i(t)$ does not satisfy $x_i(t) \leq m, \forall t \in (T_s, T_e)$. In order to prove the necessary condition, we divide $x_i(t)$ into two cases.

(a) If $x_i(t) \geq m + 1, \forall t \in (T_s, T_e)$ then let $\overline{x}(t) = m, \forall t \in (T_s, T_e)$. Then $x_i(t)$ will consume at least $(T_e - T_s)P$ more power for each extra running servers than $\overline{x}(t)$ during (T_s, T_e). On the other hand, $\overline{x}(t)$ causes at most $\beta_{on} + \beta_{off}$ more wear-and-tear cost than $x_i(t)$ for turning off/on each server. Because $(T_e - T_s) > \Delta$, $x_i(t)$ actually costs more than $\overline{x}(t)$, which is a contradiction with that $x_i(t)$ is an optimal solution.

(b) Otherwise, if $x_i(t)$ does not satisfy case (a), then there must exist time τ in (T_s, T_e) such that $x_i(\tau) = m$. Let $\overline{x}(t) = \min[m, x_i(t)], \forall t \in (T_s, T_e)$. Then $\overline{x}(t)$ satisfies all the constraints of $\mathcal{P}(\bar{a}, X, Y, T_s, T_e)$. Moreover, $\overline{x}(t)$ does not incur more on-off cost or operating cost than $x_i(t)$, which means $\overline{x}(t)$ is an optimal solution. ■

Lemma 10. *Let $\bar{x}^*(t)$ be an optimal solution to $\mathcal{P}[\bar{a}, \bar{a}(-2\Delta), \bar{a}(T + 2\Delta), -2\Delta, T + 2\Delta]$, where $\bar{a}(t)$ is defined in section III. Then $\bar{x}^*(t) = 0, \forall t \in (T, T + 2\Delta) \cup (-2\Delta, 0)$, $\bar{x}^*(T) = \bar{a}(T) = a(T)$ and $\bar{x}^*(0) = \bar{a}(0) = a(0)$. Moreover, $\bar{x}^*(t), t \in [0, T]$ is an optimal solution to SCP problem.*

Proof: Applying lemma 9, we have that $\bar{x}^*(t) = 0, \forall t \in (-2\Delta, 0)$.

Next, we prove $\bar{x}^*(0) = \bar{a}(0) = a(0)$. Assume instead that $\bar{x}^*(0) > \bar{a}(0)$. Let $\mu = \inf\{t > 0 : \bar{a}(t) \neq \bar{a}(0)\}$ be the first discontinuity in $\bar{a}$. If $\bar{a}(\mu) = \bar{a}(0)$ then let

$$\hat{x}(t) = \begin{cases} \bar{a}(0) & \forall t \in [0, \mu] \\ \bar{x}^*(t) & \text{otherwise} \end{cases}$$

Otherwise, let

$$\hat{x}(t) = \begin{cases} \bar{a}(0) & \forall t \in [0, \mu) \\ \bar{x}^*(t) & \text{otherwise.} \end{cases}$$

Since $\bar{x}^*(t) = 0, \forall t \in (-2\Delta, 0)$, $\hat{x}(t)$ incurs a lower running cost, and no higher switching cost. This contracts the assumption that $\bar{x}^*(t)$ is an optimal solution and so $\bar{x}^*(0) \leq \bar{a}(0)$. Since $\bar{x}^*(0) \geq \bar{a}(0)$ for feasibility, we have $\bar{x}^*(0) = \bar{a}(0)$. By the definition of $\bar{a}(t)$, we have $\bar{x}^*(0) = \bar{a}(0) = a(0)$.

Similarly, $\bar{x}^*(t) = 0, \forall t \in (T, T + 2\Delta)$ and $\bar{x}^*(T) = \bar{a}(T) = a(T)$.

Since $\bar{x}^*(t) = 0, \forall t \in (T, T + 2\Delta) \cup (-2\Delta, 0)$, $\bar{x}^*(T) = \bar{a}(T) = a(T)$ and $\bar{x}^*(0) = \bar{a}(0) = a(0)$. Suppose that $\bar{x}^*(t), t \in [0, T]$ is not an optimal solution to SCP. Let $x^*(t)$ denote an optimal solution to SCP. Then we let

$$\hat{x}(t) = \begin{cases} x^*(t) & \forall t \in [0, T] \\ \bar{x}^*(t) & \text{otherwise.} \end{cases}$$

Then $\hat{x}(t)$ incurs less cost than $\bar{x}^*(t)$ in $[0, T]$ and they have the same cost in the rest periods. This contracts the optimality of $\bar{x}^*(t)$ for $\mathcal{P}[\bar{a}, \bar{a}(-2\Delta), \bar{a}(T + 2\Delta), -2\Delta, T + 2\Delta]$. Therefore $\bar{x}^*(t), t \in [0, T]$ is an optimal solution to SCP. ∎

Next, we are going to prove theorem 1.

Proof: For any $\mu \in [0, T]$, we must have that μ is in some interval (τ, τ') such that $\bar{a}(\tau) \geq \bar{a}(\mu) + 1$, $\bar{a}(\tau') \geq \bar{a}(\mu) + 1$ and $\bar{a}(t) \leq \bar{a}(\mu), \forall t \in (\tau, \tau')$. We divide the situation in two cases.

Case I: $\tau' - \tau > \Delta$

In this case, according to our Optimal Solution Construction Procedure, we will set $x(\mu) = \bar{a}(\mu) = a(\mu)$.

On the other hand, according to lemma (9), $\bar{x}^*(t) \leq a(\mu), \forall t \in (\tau, \tau')$ because $\bar{x}^*(t)$ is an optimal solution to $\mathcal{P}[\bar{a}(t), \bar{a}(-2\Delta), \bar{a}(T + 2\Delta), -2\Delta, T + 2\Delta]$. Therefore, we must have $\bar{x}^*(\mu) = \bar{a}(\mu) = a(\mu)$. This means in Case I our Optimal Solution Construction Procedure gives an optimal solution.

Case II: $\tau' - \tau \leq \Delta$

In this case, since $\tau' - \tau \leq \Delta$, we must have that $\tau, \tau' \in [0, T]$. Therefore, there must exist two intervals $\left(\tau_1, \tau_1'\right)$ and $\left(\tau_2, \tau_2'\right)$ which have following properties:

(1) $\left(\tau_1, \tau_1'\right) \supseteq (\tau, \tau')$, $\bar{a}(\tau_1) \geq \bar{a}(\mu) + 1$, $\bar{a}\left(\tau_1'\right) \geq \bar{a}(\mu) + 1$ and $\tau_1' - \tau_1 \leq \Delta$. Moreover, for any interval $\left(\upsilon_1, \upsilon_1^1\right) \supseteq \left(\tau_1, \tau_1'\right)$ with $\bar{a}(\upsilon_1) \geq \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right] + 1$ and $\bar{a}\left(\upsilon_1'\right) \geq \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right] + 1$, we must have $\upsilon_1' - \upsilon_1 > \Delta$.

(2) $\left(\tau_2, \tau_2'\right) \supseteq \left(\tau_1, \tau_1'\right)$, $\bar{a}(\tau_2) \geq \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right] + 1$, $\bar{a}\left(\tau_2'\right) \geq \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right] + 1$, $\tau_2' - \tau_2 > \Delta$ and $a(t) \leq \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right], \forall t \in \left(\tau_2, \tau_2'\right)$.

In this case, according to our Optimal Solution Construction Procedure, we will set $x(\mu) = \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right]$.

On the other hand, according to lemma (9), $\bar{x}^*(t) \leq \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right], \forall t \in \left(\tau_2, \tau_2'\right)$ because $\bar{x}^*(t)$ is an optimal solution to $\mathcal{P}[\bar{a}(t), \bar{a}(-2\Delta), \bar{a}(T + 2\Delta), -2\Delta, T + 2\Delta]$. Then $\bar{x}^*(t) \geq \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right]$, $\forall t \in \left(\tau_1, \tau_1'\right)$, because turning a server off and on later incurs no less cost than just letting it be idle during $\left(\tau_1, \tau_1'\right)$ since $\tau_1' - \tau_1 \leq \Delta$. Therefore $\bar{x}^*(\mu) = \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right]$. This means in Case II our Optimal Solution Construction Procedure also gives an optimal solution.

Thus the $x(t)$ constructed by the Optimal Solution Construction Procedure is an optimal solution to $\mathcal{P}[\bar{a}(t), \bar{a}(-2\Delta), \bar{a}(T + 2\Delta), -2\Delta, T + 2\Delta]$, and the result follows by lemma 10. ∎

B. Proof of Theorem 2

First we are going to prove that the offline algorithm solves $\mathcal{P}[\bar{a}(t), \bar{a}(-2\Delta), \bar{a}(T + 2\Delta), -2\Delta, T + 2\Delta]$. Lemma 10 then implies the offline algorithm also solves SCP. First, we are going to prove following lemma.

Lemma 11. *Under last-empty-server-first job dispatching, if a server becomes empty at τ_1 and it will receive the first job after τ_1 at τ_1', then $\bar{a}(\tau_1) = \bar{a}\left(\tau_1'\right)$ and $\bar{a}(t) < \bar{a}(\tau_1), \forall t \in \left(\tau_1, \tau_1'\right)$.*

Proof: Let $\mathcal{S}$ be the ID of the server becoming empty at τ_1. Since τ_1' is the first time that $\mathcal{S}$ is popped after τ_1, the number of server IDs below $\mathcal{S}$ on the stack does not change during $\left(\tau_1, \tau_1'\right)$. At any time, the number of jobs in the system is equal to the number of server's IDs that are not in the stack of the job-dispatching entity. Therefore $\bar{a}(\tau_1) = \bar{a}\left(\tau_1'\right)$ and $\bar{a}(t) < \bar{a}(\tau_1), \forall t \in \left(\tau_1, \tau_1'\right)$. ∎

Lemma 12. *For any idle server at time μ, let τ be the most recent time before μ that the server became idle. Then $\bar{a}(\tau) > \bar{a}(\mu)$.*

Proof: Let τ' be the first time after μ that the server receives a job. By lemma 11, we have $\bar{a}(t) < \bar{a}(\tau), \forall t \in (\tau, \tau')$. Thus $\bar{a}(\tau) > \bar{a}(\mu)$. ∎

Now, we are going to prove theorem 2. The proof is similar to the proof of theorem 1. Let $x_o(t)$ Denote the number of servers run by the offline algorithm at t.

Proof: For any $\mu \in [0, T]$, we must have that μ is in some interval (τ, τ') such that $\bar{a}(\tau) \geq \bar{a}(\mu) + 1$, $\bar{a}(\tau') \geq \bar{a}(\mu) + 1$ and $\bar{a}(t) \leq \bar{a}(\mu), \forall t \in (\tau, \tau')$. We divide the situation in two cases.

Case I: $\tau' - \tau > \Delta$

In this case, according to our Optimal Solution Construction Procedure, we will set $x(\mu) = \bar{a}(\mu) = a(\mu)$.

We are going to prove that there is no idle server in the system at μ if we are running the proposed offline algorithm. We will divide the situation in three sub-cases.

(1) $\tau = -2\Delta$

In this sub-case, if there are idle servers at μ, then some idle servers will receive jobs at τ'. According to lemma 11, there must exist a time υ in $[0, \tau')$ such that $\bar{a}(\upsilon) > \bar{a}(\mu)$, which contradicts that $\bar{a}(t) \leq \bar{a}(\mu), \forall t \in (-2\Delta, \tau')$.

(2) $\tau' = T + 2\Delta$

In this sub-case, if there are idle servers at μ, by lemmas 11 and 12, there must exist a time υ in (τ, T) such that $\bar{a}(\nu) > \bar{a}(\mu)$, which contradicts that $\bar{a}(t) \leq \bar{a}(\mu), \forall t \in (\tau, T + 2\Delta)$.

(3) $(\tau, \tau') \in [0, T]$

In this sub-case, if there are idle servers at μ, by lemmas 11 and 12, the idle server will receive a job after τ'. Thus the idle period for the idle server is larger than $\tau' - \tau > \Delta$, which contradicts the fact that in our offline algorithm no server is longer than Δ.

The three sub-cases shows that in Case I there is no idle server at μ, which means $x_o(\mu) = x(\mu)$. Therefore, the offline algorithm gives an optimal solution to $\mathcal{P}[\bar{a}(t), \bar{a}(-2\Delta), \bar{a}(T + 2\Delta), -2\Delta, T + 2\Delta]$.

Case II: $\tau' - \tau \le \Delta$

In this case, since $\tau' - \tau \le \Delta$, we must have that $\tau, \tau' \in [0, T]$. Therefore, there must exist two intervals $\left(\tau_1, \tau_1'\right)$ and $\left(\tau_2, \tau_2'\right)$ which have following properties:

(1) $\left(\tau_1, \tau_1'\right) \supseteq (\tau, \tau')$, $\bar{a}(\tau_1) \ge \bar{a}(\mu) + 1$, $\bar{a}\left(\tau_1'\right) \ge \bar{a}(\mu) + 1$ and $\tau_1' - \tau_1 \le \Delta$. Moreover, for any interval $\left(\upsilon_1, \upsilon_1^1\right) \supseteq \left(\tau_1, \tau_1'\right)$ with $\bar{a}(\upsilon_1) \ge \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right] + 1$ and $\bar{a}\left(\upsilon_1'\right) \ge \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right] + 1$, we must have $\upsilon_1' - \upsilon_1 > \Delta$.

(2) $\left(\tau_2, \tau_2'\right) \supseteq \left(\tau_1, \tau_1'\right)$, $\bar{a}(\tau_2) \ge \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right] + 1$, $\bar{a}\left(\tau_2'\right) \ge \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right] + 1$, $\tau_2' - \tau_2 > \Delta$ and $a(t) \le \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right], \forall t \in \left(\tau_2, \tau_2'\right)$.

In this case, according to our Optimal Solution Construction Procedure, we will set $x(\mu) = \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right]$.

Similarly to Case I, we can also divide the situation into three sub-cases: (1) $\tau_2 = -2\Delta$. (2) $\tau_2' = T + 2\Delta$. (3) $\left(\tau_2, \tau_2'\right) \in [0, T]$. In each sub-case, we can adopt the approach we used in Case I to show that $x_o(t) = \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right], \forall t \in (\tau_2, \tau_1)$. By lemma 11, the offline algorithm will not turn off a server during $\left(\tau_1, \tau_1'\right)$. Therefore $x_o(\mu) = \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1'\right)\right] = x(\mu)$.

In the two cases, we proved that $x_o(t)$ is equal to $x(t)$ constructed by Optimal Solution Construction Procedure. Therefore, the proposed offline algorithm solves SCP optimally. ■

C. Least idle vs last empty

In order to prove that least-idle is at least as good as last-empty, we are going to prove two facts: (i) the number of physical switches in least-idle is no more than last-empty, and (ii) the number of "on" servers at any given time under least-idle is also no more than that of last-empty.

Let $L(s,t)$ be a time-varying permutation of servers such that any arrival to or departure from server s at time t under last-empty arrives to or departs from $L(s,t)$ under least-idle. (Note this is a random variable depending on the random variables Z chosen at times prior to t.) Multiple such permutations exist; we impose the continuity condition so that $L(s,t)$ only changes at job arrival times. Specifically, if $L(s_1,t_1) = L(s_2,t_2)$ for $s_1 \ne s_2$ and $t_1 < t_2$ then there is an arrival to either s_1 or s_2 under last-empty in the interval $[t_1,t_2]$, and least-idle assigns the job to a different server.

Partition the interval $[0,T)$ as follows. Let $\mathcal{D}_s$ be the set of points of discontinuity of $L(s,\cdot)$. We claim that, with probability 1, there are no accumulation points in $\mathcal{D}_s$. To see this, note that an accumulation point would only occur if there were an interval of length ϵ such that there were an infinite number of (i.i.d.) random timeouts Z generated, each of which is less than ϵ. We can then partition $[0,T)$ into intervals of the form $[a_s(i), a_s(i+1))$, where $a_s(\cdot) \in \mathcal{D}_s$. According to the continuity condition of $L(s,t)$, all the points $a_s(\cdot)$ in $\mathcal{D}_s$ are job arrival points.

We can think of $L(s,\cdot)$ as defining a "logical" server that serves the same jobs under least-idle as s does under last-empty. By hypothesis, it also generates the same sequence of Z sleep timeouts under least-idle as s does under last-empty. There are two ways that logical server $L(s,\cdot)$ can turn on are: (i) the mapping $L(s,\cdot)$ remains constant and the server $L(s,t)$ turns on. (ii) the mapping $L(s,\cdot)$ changes from a server that is off to a server that is on. Consequently, the only times that logical server $L(s,\cdot)$ turns on or off and server s does not are in the case (ii). These switches do not correspond to a physical server turning on or off, and so do not incur a switching cost. Hence the total switching cost under least-idle is at most that under last-empty.

It remains to show that $x_{L(s,t)}(t)$ under least-idle is at most $x_s(t)$ under last-empty. The only cause for $L(s,\cdot)$ to turn on is a new arrival, after which both s and $L(s,t)$ must be on. The only times that s turns off that $L(s,\cdot)$ does not are during idle periods when $L(s,\cdot)$ has already turned off "for free" due to a discontinuity in $L(s,\cdot)$.

D. Proof of Theorem 5

In order to prove theorem 5, we use Lemma 3 and two other technical lemmas. First, let us introduce some notation.

Let $\tau_{j,s}$ be the time in $[0,T]$ that job j arrives at server s, and $\tau_{j,e}$ be the time that j leaves the system. Let $\tau_{j,s}^{\dagger} = \inf\{t > \tau_{j,e} :$ a job arrives to $s$at time $t\}$; if there are finitely many arrivals in $[0,T]$ then $\tau_{j,s}^{\dagger} = \tau_{j+1,s}$. We also consider time T as a virtual job arrival point to the server.

Lemma 13. *The deterministic online ski-rental algorithm used by the online algorithm CSR has competitive ratio* $2 - \alpha$.

Proof: As we already proved in Lemma 3, for both online and off-line cases, a server faces the same set of jobs. From now on, we focus on one server, s. The server should decide to turn itself off or stay idle between $\tau_{j,e}$ and $\tau_{j,s}^{\dagger}$. In order to find the competitive ratio, we compare the costs P_j^{on} and P_j^{off} of the online and off-line ski-rental algorithms respectively in $\left(\tau_{j,s}, \tau_{j,s}^{\dagger}\right]$. This does not include the cost to turn on at $\tau_{j,s}$, but does include the cost to turn on at $\tau_{j,s}^{\dagger}$ (or immediately after if $\tau_{j,s}^{\dagger}$ is an accumulation point). The costs of the online and off-line ski-rental algorithms depend on the length of the time between $\tau_{j,e}$ and $\tau_{j,s}^{\dagger}$. Let $T_{j,B} = \tau_{j,e} - \tau_{j,s}$ denote the length of the busy period in $\left(\tau_{j,s}, \tau_{j,s}^{\dagger}\right]$ and $T_{j,E} = \tau_{j,s}^{\dagger} - \tau_{j,e}$ denote the length of the empty period in $\left(\tau_{j,s}, \tau_{j,s}^{\dagger}\right]$. Then

$$P_j^{off} = \begin{cases} P_b T_{j,B} + P T_{j,E} & if\ T_{j,E} \le \Delta \\ P_b T_{j,B} + \left(\beta_{on} + \beta_{off}\right) & if\ T_{j,E} > \Delta \end{cases} \tag{8}$$

and the online ski-rental algorithm in CSR gives

$$P_j^{on} = \begin{cases} P_b T_{j,B} + P T_{j,E} & if\ T_{j,E} \le \Delta \\ P_b T_{j,B} + \left(\beta_{on} + \beta_{off}\right) + P(1-\alpha)\Delta & if\ T_{j,E} > \Delta \end{cases} \tag{9}$$

Hence, $T_{j,E} \le \Delta$ implies $P_j^{on}/P_j^{off} = 1$, and since $P\Delta = \left(\beta_{on} + \beta_{off}\right)$, $T_{j,E} > \Delta$ implies

$$\frac{P_j^{on}}{P_j^{off}} \le \frac{\left(\beta_{on} + \beta_{off}\right) + P(1-\alpha)\Delta}{\left(\beta_{on} + \beta_{off}\right)} = 2 - \alpha.$$

In either case, $P_j^{on}/P_j^{off} \le 2 - \alpha$ for any $T_{j,E}$. Summing over j and s gives the result. ■

Lemma 14. *The randomized online ski-rental algorithm used by the online algorithm RCSR with last-empty-server-first strategy has competitive ratio $e/(e-1+\alpha)$.*

Proof: In the proof, we again focus on one server. We will use the notation used to prove Lemma 13. This time it is sufficient to compare the average cost P_j^{on} of the randomized online ski-rental algorithm in $\left(\tau_{j,s}, \tau_{j,s}^{\dagger}\right]$ with off-line optimal cost in (8). Under the randomized online ski-rental algorithm, when $T < \alpha\Delta$, we have

$$\mathbb{E}\left(P_j^{on}\right) = P_b T_{j,B} + P T_{j,E};$$

when $\alpha\Delta \le T_{j,E} \le \Delta$, we have

$$\begin{aligned}\mathbb{E}\left(P_j^{on}\right) &= P_b T_{j,B} + \int_0^{T_{j,E}-\alpha\Delta} \left(Pz + \beta_{on} + \beta_{off}\right) f_Z(z)\, dz \\ &\quad + P \int_{T_{j,E}-\alpha\Delta}^{(1-\alpha)\Delta} T_{j,E} f_Z(z)\, dz;\end{aligned}$$

and when $T_{j,E} > \Delta$, we have

$$\mathbb{E}\left(P_j^{on}\right) = P_b T_{j,B} + \int_0^{(1-\alpha)\Delta} \left(Pz + \beta_{on} + \beta_{off}\right) f_Z(z)\, dz.$$

We get the above expected cost for $\alpha\Delta \le T_{j,E} \le \Delta$ as follows: If the number Z generated by the server is less than $T_{j,E} - \alpha\Delta$, then the server will wait for time Z, consuming energy PZ. It looks into the look-ahead window of size $\alpha\Delta$ and finds it won't receive any job during the window because $Z < T_{j,E} - \alpha\Delta$. Therefore, it turns itself off and costs $\left(\beta_{on} + \beta_{off}\right)$. On the other hand, if $Z \ge T_{j,E} - \alpha\Delta$, the server will not turn itself off and consume $PT_{j,E}$ to stay idle. We can get the expected cost for $T_{j,E} < \alpha\Delta$ and $T_{j,E} > \Delta$ in the same way. According to the distribution of Z in RCSR, we can calculate $\mathbb{E}\left(P_{j,on}\right)$ and the ratio between $\mathbb{E}\left(P_{j,on}\right)$ and $P_{j,off}$:

$$\frac{\mathbb{E}\left(P_j^{on}\right)}{P_j^{off}} = \begin{cases} 1, & T_{j,E} < \alpha\Delta \\ \frac{e}{e-1+\alpha} & T_{j,E} \ge \alpha\Delta \end{cases}$$

From this expression, for all j, we can conclude that $\mathbb{E}\left(P_j^{on}\right)/P_j^{off} \le \frac{e}{e-1+\alpha}$ for any $T_{j,E}$. Summing over j and s gives the result. ∎

Now we are ready to prove theorem *5*.

Recall that the optimal cost of the data center can be achieved by each server running an off-line ski-rental algorithm independently. On the other hand, in Lemmas 13 and 14, we proved that the cost of deterministic and randomized online ski-rental algorithm we applied are at most $2-\alpha$ and $\frac{e}{e-1+\alpha}$ times the cost of off-line ski-rental algorithm for one server. Therefore, the cost of our online algorithm CSR is at most $2-\alpha$ times the cost of off-line algorithm for data center. Moreover, if we adopt last-empty-server-first job-dispatching strategy in the randomized algorithm RCSR, it can achieve competitive ratio $\frac{e}{e-1+\alpha}$. By Lemma 4, RCSR with least-idle performs at least as well as if it adopted last-empty-server-first job-dispatching strategy. Therefore, RCSR has competitive ratio $\frac{e}{e-1+\alpha}$.

Next, we want to prove that CSR has the best competitive ratio for deterministic online algorithms. First, we prove that the best competitive ratio of a deterministic algorithm for a single ski-rental problem is $2-\alpha$. Assume that the deterministic online algorithm peeks into the look-ahead window and then decide to turn off or stay idle time $\theta\Delta$ after becoming empty at t_1. For $\theta < 1-\alpha$, if the server receives its next job right after $t_1 + (\theta+\alpha)\Delta$, then the online algorithm will turn off itself at $t_1 + \theta\Delta$, and consume energy $P(\theta+1)\Delta$. On the other hand, the offline optimal is $(\alpha+\theta)P\Delta$, whence the competitive ratio is at least $\frac{\theta+1}{\theta+\alpha} > 2-\alpha$. For $\theta > 1-\alpha$, if the server receives its next job right after $t_1 + (\theta+\alpha)\Delta$, then the online algorithm will turn off itself at $t_1 + \theta\Delta$, and consume $P(\theta+1)\Delta$ power. On the other hand, the offline optimal is $P\Delta$. The competitive ratio at least is $1 + \theta > 2-\alpha$. Hence, only when $\theta = 1-\alpha$ can the deterministic algorithm have the competitive ratio $2-\alpha$. Therefore, the best competitive ratio of deterministic algorithm for a single ski-rental problem is $2-\alpha$. However, a server will receive a sequence of jobs in data center. And after finishing each job, the server faces a ski-rental problem. Hence, each server actually faces a repeated ski-rental problem. As for repeated ski-rental problem we have following lemma.

Lemma 15. *The best competitive ratio of a deterministic algorithm for the repeated ski-rental problem faced by each server is $2-\alpha$.*

Proof: We will prove lemma 15 by induction. Assume that deterministic algorithm $A1$ achieves the best competitive ratio for repeated ski-rental problem. Let $\theta_i\Delta$ be the length of idle time before the server peeks into the look-ahead in the ith ski-rental problem. Since the best deterministic algorithm for single ski-rental problem must peek into the look-ahead window after staying idle for $(1-\alpha)\Delta$, $A1$ must have $\theta_1 = 1-\alpha$.

Suppose $\theta_i = 1-\alpha$ for $i = 1, 2, ...k$. Therefore, the cost of $A1$ is up to $2-\alpha$ times the off-line optimal for the first k ski-rental problems. We will prove that $A1$ must have $\theta_{k+1} = 1-\alpha$. If $\theta_{k+1} < 1-\alpha$ or $\theta_{1+k} > 1-\alpha$, we can use the same approach we used to prove the best competitive ratio for single ski-rental is $2-\alpha$ to show that in the $(k+1)$th ski-rental problem $A1$ consumes more than $2-\alpha$ times the off-line optimal in worst case for this phase. If the worst case of this phase occurs after the worst case of the previous phases, this would result in an overall competitive ratio exceeding $2-\alpha$. Therefore, we must have $\theta_{k+1} = 1-\alpha$ if the competitive ratio is to be at most $2-\alpha$. Since the scheme with $\theta_k = 1-\alpha$ for all k has competitive ratio exactly $2-\alpha$, it follows that the best competitive ratio of deterministic algorithm for repeated ski-rental algorithm is $2-\alpha$. ∎

When $a(t) \le 1$, the problem SCP we study in this paper becomes a repeated ski-rental problem. Therefore, the best competitive ratio for a deterministic online algorithm is $2-\alpha$, which is achieved by CSR.

Finally, we want to prove that RCSR has the best competitive ratio for randomized online algorithms. Consider the case that the server becomes empty at τ_1 and it will receive its next job at τ_2. In order to find the best competitive ratio for a randomized online algorithm, according to the proof of Lemma 14, it is sufficient to find the minimal ratio of the cost by a randomized online algorithm to that of the offline

optimal in $[\tau_1, \tau_2]$. The competitive ratio cannot be lower than the competitive ratio on an instance with a single empty interval, and so we consider that case. We first divide time period (τ_1, τ_2) into slots of equal length. As the length of the slots goes to zero, we can get the best competitive ratio for a continuous time randomized online algorithm.

Assume the critical interval Δ contains exact b slots and there are D slots in $[\tau_1, \tau_2]$. We focus on the case that the look-ahead window has $k \leq b-2$ slots. (If $k \geq b-1$, the online algorithm can achieve the offline optimum and the competitive ratio is 1.) Let p_i denote the probability that the algorithm decides to turn off the server at slot $i = 1, 2, \ldots$. Let the competitive ratio be c. Regardless of the value of D, the expected online cost must be at most the competitive ratio times the off-line cost. Thus the minimum competitive ratio satisfies

$$\inf \quad c \tag{10}$$

$$\text{s.t.} \quad D\sum_{i=1}^{\infty} p_i \leq cD, \quad \forall D \in [0,k], \tag{11}$$

$$\sum_{i=1}^{D-k} (b+i-1)\, p_i + \sum_{i=D-k+1}^{\infty} D p_i \leq Dc, \ \forall D \in (k,b] \tag{12}$$

$$\sum_{i=1}^{D-k} (b+i-1)\, p_i + \sum_{i=D-k+1}^{\infty} D p_i \leq bc, \ \forall D \in (b,\infty] \tag{13}$$

$$\sum_{i=1}^{\infty} p_i = 1, \qquad 0 \leq p_i \leq 1, \forall i \tag{14}$$

$$\text{var} \quad c, p_i, \quad \forall i \in \{1,2,3,\ldots\} \tag{15}$$

We can apply the steps in [36] to show that the optimal value c_d^* of problem (10)–(15) is equal to the optimal value $\bar{c}^*$ of following problem.

$$\min \quad \bar{c} \tag{16}$$

$$\text{s.t.} \quad 1 \leq \bar{c}, \quad \forall D \in [0,k], \tag{17}$$

$$\sum_{i=1}^{D-k} (b+i-1)\, \bar{p}_i + \sum_{i=D-k+1}^{b-k} D\bar{p}_i \leq D\bar{c}, \ \forall D \in (k,b) \tag{18}$$

$$\sum_{i=1}^{b-k} (b+i-1)\, \bar{p}_i \leq b\bar{c}, \quad \forall D \in [b,\infty] \tag{19}$$

$$\sum_{i=1}^{b-k} \bar{p}_i = 1 \qquad 0 \leq p_i \leq 1, \forall i \tag{20}$$

$$\text{var} \quad \bar{c}, \bar{p}_i, \quad \forall i \in \{1,2,\ldots,b-k\} \tag{21}$$

Next, we prove that $\bar{p}_1^*$ is positive. If instead $\bar{p}_1^* = 0$, let j be the minimal i such that $\bar{p}_i^* > 0$. Then the constraints (18)–(19) must hold as strict inequalities for $D \leq k+j-1$, for the following reason. First consider the constraint for $D = k+j$. Since we have $j \leq b-k-1$ (otherwise we obtain the deterministic algorithm CSR, which is suboptimal), we have $D = k+j < b$ and the constraint for $D = k+j$, divided by D, is

$$\frac{b+j-1}{k+j}\bar{p}_j + \sum_{i=j+1}^{b-k} \bar{p}_i \leq \bar{c}$$

and when $D \leq k+j-1$, the constraints, divided by D, are

$$\sum_{i=j}^{b-k} \bar{p}_i \leq \bar{c}.$$

Since $k \leq b-2$, if the latter were active, then the former would be violated.

We use the slackness of these constraints to show $\bar{p}_1^* > 0$. The coefficient of $\bar{p}_1^*$ is less than that of $\bar{p}_j^*$ in the constraints for $D > k+j-1$. Therefore, we can decrease $\bar{p}_j^*$ a little bit and increase $\bar{p}_1^*$ a little bit such that all the constraints of (17)–(20) have slackness, which means we can find a smaller $\bar{c}$ which satisfies all the constraints. This contradicts the optimality of $\bar{p}^* = \left[\bar{p}_1^*, \bar{p}_2^*, \bar{p}_3^*, \cdots, \bar{p}_{b-k}^*\right]$. Therefore, we must have $\bar{p}_1^* > 0$.

Next, we again follow [36] to show that each of the inequalities in (18), (19) is tight. Assume instead that the constraint corresponding to some particular $D \in (k,b]$ is loose. Let $D^{\#}$ be the largest such D. Consider case (i) that $D^{\#} < b$. Note that $\bar{p}^*_{D^{\#}-k+1} > 0$, since otherwise $D^{\#}+1$ would also be slack. Then decrease $\bar{p}^*_{D^{\#}-k+1}$ and increase $\bar{p}^*_{D^{\#}-k}$ slightly. This does not affect constraints for smaller D, but introduces slack into the constraints for all larger D. Next, we could increase $\bar{p}^*_{D^{\#}-k}$ and decrease $\bar{p}_1^*$, which doesn't affect constraints for larger D, but introduces slack for all constraints with smaller D. Alternatively, in case (ii) that $D^{\#} = b$, we can decrease $\bar{p}_1^*$ while increasing $\bar{p}^*_{b-k}$ to introduce slack into earlier constraints. In either case, the transformation induces slack in all constraints, which allows $\bar{c}^*$ to decrease, contradicting the optimality of $\bar{c}^*$. Therefore, all the constraints for $D \in (k,b]$ must be tight.

Since the total $b-k$ constraints for all the $D \in (k,b]$ is tight and $\sum_{i=1}^{b-k} \bar{p}_i = 1$, we can solve the system of linear equations and get the minimal competitive ratio and probability distribution:

$$\bar{c}^* = \left(1 - \left(\frac{b-k-1}{b-k}\right)^{b-k-1} \frac{b-k-1}{b}\right)^{-1}$$

$$\bar{p}^*_{b-k-i} = \frac{\bar{c}^*}{b-k}\left(\frac{b-k-1}{b-k}\right)^{i}, \qquad 0 \leq i < b-k-1$$

$$\bar{p}_1^* = \left(\frac{b-k-1}{b-k}\right)^{b-k-1} \frac{k+1}{b}\bar{c}^*, \qquad k < b$$

Letting b go to infinity and keeping $k/b = \alpha$, we have the minimal competitive ratio c^* for continuous time:

$$c^* = \frac{e}{e-1+\alpha}$$

This means the optimal competitive ratio for continuous time randomized online algorithm is $c^* = \frac{e}{e-1+\alpha}$, as required. Therefore, the best competitive ratio of randomized algorithm for single ski-rental problem is $\frac{e}{e-1+\alpha}$. We have the following lemma to prove that RCSR has the best competitive ratio for randomized algorithms against oblivious adversary.

Lemma 16. *The best competitive ratio of randomized algorithm for the repeated ski-rental problem is* $\frac{e}{e-1+\alpha}$.

Proof: In the proof we will use the notation used in the proof of lemma 13. Assume that the cost of online algorithm

in the ith ski rental is C_i and the corresponding offline optimal is C_i^*. If the strategy of the oblivious adversary is to arbitrarily choose a number which is greater than $\alpha\Delta$ as the empty period of each ski rental problem, then the online algorithm has no information of the length of current empty period $T_{i,E}$ even if the online algorithm knows $T_{i-1,E}$, $T_{i-2,E}$,...$T_{1,E}$. We are going to prove lemma 16 by induction.

It is clear that in the first ski rental we can not do better than in single ski rental problem. Therefore, we have $E(C_1) \leq \frac{e}{e-1+\alpha} C_1^*$.

Assume that $E\left(\sum_{i=1}^{k-1} C_i\right) \leq \frac{e}{e-1+\alpha} \sum_{i=1}^{k-1} C_i^*$, we are going to prove $E\left(\sum_{i=1}^{k} C_i\right) \leq \frac{e}{e-1+\alpha} \sum_{i=1}^{k} C_i^*$.

Suppose online algorithm chooses $f_{Z_k}(z_k|z_{k-1}, z_{k-2}...z_1)$ as the conditional probability distribution of Z_k given the historical information. Then there always exists a $\bar{T}_{k,E}$ such that

$$\begin{aligned} E(C_k) &= E(E(C_k|Z_{k-1}, Z_{k-2}...Z_1)) \\ &\geq \frac{e}{e-1+\alpha} C_k^* \end{aligned}$$

To see this, suppose there is no $\bar{T}_{k,E}$ satisfying above inequality, then for any $T_{k,E}$, we must have

$$E(E(C_k|Z_{k-1}, Z_{k-2}...Z_1)) < \frac{e}{e-1+\alpha} C_k^*$$

Then in the single ski rental problem, we can also let the distribution of random variable Z follow the unconditional distribution $f_{Z_k}(z_k)$ of Z_k. In this way, we can get a better competitive ratio than $\frac{e}{e-1+\alpha}$ for single ski rental problem. This is a contradiction. Therefore, such $\bar{T}_{k,E}$ must exist. This means the best ratio we can do in kth ski rental is $\frac{e}{e-1+\alpha}$. Since $E\left(\sum_{i=1}^{k} C_i\right) = E\left(\sum_{i=1}^{k-1} C_i\right) + E(C_k)$ and $\sum_{i=1}^{k} C_i^* = \sum_{i=1}^{k-1} C_i^* + C_k^*$, thus we have

$$E\left(\sum_{i=1}^{k} C_i\right) \leq \frac{e}{e-1+\alpha} \sum_{i=1}^{k} C_i^*$$

as required. This means online algorithms can not do better than $\frac{e}{e-1+\alpha}$ even against oblivious adversary. Therefore, RCSR has the best competitive ratio $\frac{e}{e-1+\alpha}$ for randomized algorithms against oblivious adversary. Since SCP has repeated ski rental problem as a special case, $\frac{e}{e-1+\alpha}$ is the optimal competitive ratio for any randomized algorithm. ∎

E. Proof of Corollary 7

In this section, we are going to prove corollary 7.

Proof: We prove the result for RCSR. Since we make no use of the form of f_X, the same proof holds for CSR (which corresponds to RCSR with $f_x(x) = \delta(x - \Delta)$).

We first establish validity. Note that $x(t)$ is the number of servers ON under RCSR, and that x increases by at most a factor of $1 + \gamma$ in an interval of length T_s, since $x(t) \geq a(t)$ at the start, and $x(t) = a(t)$ at all times that x increases. Since arrival instants are discrete, there are also no limit point in the set of times $t_n{}_{n=0}^{N}$ at which x changes, and so we can apply induction on n.

By induction, the number of ON and BOOT servers at each time t_n is either $\lceil x(t_n)(1+\gamma) \rceil$ or $\lceil x(t_n)(1+\gamma) \rceil + 1$. The base case, $t_0 = 0$, is true by hypothesis. For subsequent t_n it is true by construction except that when M is sent, there may be only $\lfloor x(t_{n-1})(1+\gamma) \rfloor + 1 \geq \lceil x(t_n)(1+\gamma) \rceil$ servers ON or BOOTing.

We now show by induction that there are at least $x(t_n)$ ON servers at each time t_n. If x decreases at t_n, this is true since there were at least $x(t_{n-1}) > x(t_n)$ servers ON before t_n. Next consider the case that x increases at t_n.

Let $\tau = \arg\min_{\tau \in [t_n - T_s, t_n]} x(\tau)$, with ties broken by taking the smallest τ. We claim all BOOT servers at τ were BOOT at $t_n - T_s$. This is trivial if $\tau = t_n - T_s$. To see it in other cases, suppose instead there is a BOOT server at τ that was turned on at $\tau' \in (t_n - T_s, \tau)$. Now $x(\tau') > x(\tau)$ by the minimality of τ, and so $\lceil x(\tau')(1+\gamma) \rceil \geq \lceil x(\tau)(1+\gamma) \rceil + 1$, whence there are more ON or BOOT servers at τ' than at τ. However, since EXT turns of the most recently turned on BOOT servers first, existence at τ of a BOOT server turned on at τ' means that more servers are turned on during $[\tau', \tau]$ than are turned off, which is a contradiction.

Since $x(t_n) \leq x(\tau)(1+\gamma)$ and all the BOOT servers at τ will become ON at t_n, thus there will be at least $x(\tau) + \lfloor x(\tau)\gamma\% \rfloor + 1 \geq x(t_n)$ ON servers. This completes the induction.

Since $x(t) \geq a(t)$, we have proved the number of ON servers is at least $a(t)$ at time t in the extended algorithm, which establishes the first claim of the theorem.

To prove the competitive ratios, note that the number of total active servers in EXT is at most $(1+\gamma)x(t) + 2$. The total running energy cost of EXT is at most $2PT$ more than $(1+\gamma)$ times the running cost of RCSR.

Now, we are going to analyze the switching cost. We divide the $x(t)$ down into periods during which $x(t)$ is increasing and periods in which it is decreasing. Moreover, a decreasing/increasing period must be followed by a increasing/decreasing period and the combination of all the periods covers the interval of $[0, T]$. In any increasing period, assume that $x(t)$ increase from A to $A + k$, the number of turning-on in extended algorithm is

$$\lfloor (A+k)(1+\gamma) \rfloor - \lceil A(1+\gamma) \rceil \leq k(1+\gamma)$$

We will get similar result for decreasing period. Therefore, the total switching cost of the extended algorithm is at most $(1+\gamma)$ times that of RCSR.

When servers have setup time, the offline optimal cost P_S^* is changed. However, the optimal value of SCP is a lower bound of P_S^*. Hence, the total cost of RCSR is at most $\frac{e}{e-1+\alpha} P_S^*$. Moreover, the total cost of EXT is at most $2PT + \frac{e}{e-1+\alpha}(1+\gamma) P_S^*$. The competitive ratio of EXT follows from that a_{min} is the minimal workload. ∎

F. Experiment of Elephant Workload

In this section, we will evaluate CSR and RCSR with "elephant" workload defined in Section II-A. Since we do not have any real data center trace of this kind of workload, we used computer to generate a synthetic workload shown in Fig. 5. In this "elephant" workload trace, the job arrival rate is

Poisson process and the rate varies from an hour to another. The service time of each job is exponentially distributed and the mean is about 33 hour. Since we need use computer to do simulation, we chop the total time in to small slots with length of 6 seconds. The PMR of this trace is 1.7. The simulation result is shown in Fig. 6. One observation from Fig. 6 is that the CSR and RCSR energy-saving curves of "elephant" workload are similar to that of "mice" workload. Our algorithms can save more than 37% energy. And this number is consistent with the result in Fig. 4d which indicates that the energy saving is about 40% when PMR is 2.

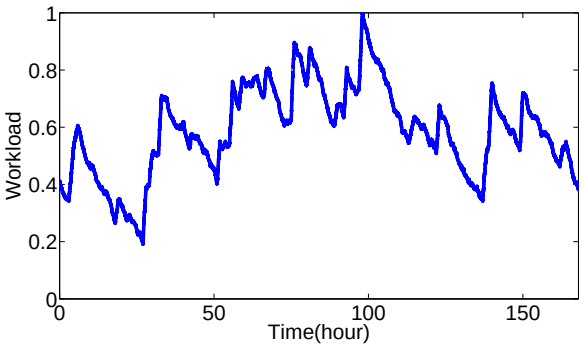


Figure 5: Synthetic "Elephant" Workload.

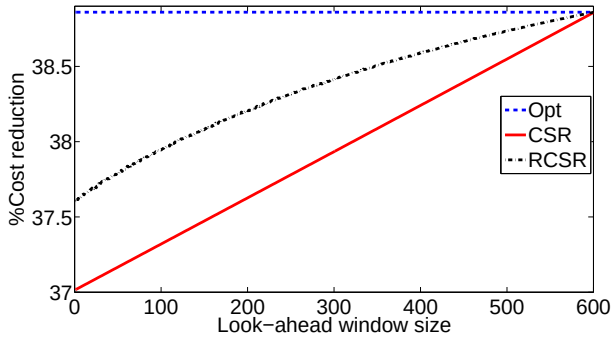


Figure 6: Energy saving of our algorithms for "elephant" workload.