Simple and Effective Dynamic Provisioning for Power-proportional Data Centers

LU, Tan

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Information Engineering

The Chinese University of Hong Kong
September 2012

Abstract of thesis entitled:

Simple and Effective Dynamic Provisioning for Power-proportional Data Centers

Submitted by LU, Tan

for the degree of Master of Philosophy

at The Chinese University of Hong Kong in September 2012

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 thesis, 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 offline 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 $e/\left(e-1+\alpha\right)$, respectively, where $0 \leq \alpha \leq 1$ 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.

When designing online algorithms, we utilize future input infor-

mation because for many modern systems, their short-term future inputs can be predicted by machine learning, time-series analysis, etc. We also test our algorithms in the presence of prediction errors in future workload information and the results show that our algorithms are robust to prediction errors. We believe that utilizing future information is a new and important degree of freedom in designing online algorithms. In traditional online algorithm design, future input information is not taken into account. Many online problems have online algorithms with optimal but large competitive ratios. Since future input information to some extent can be estimated accurately in many problems, we believe that we should exploit such information in online algorithm design to achieve better competitive ratio and provide more competitive edge in both practice and theory.

Acknowledgement

I would like to thank my supervisor Prof. Chen Minghua, whose valuable advice and consistent encouragement are highly appreciated. His attitude and spirit toward research and life will affect me for the rest of my life. I would like to show my gratitude to Prof. Lachlan Andrew for his valuable advice and great effort to make this project better. I also thank all the people in our lab for their help. It was so much fun with them. I also want to thank Minghong Lin for sharing the code of his LCP algorithm, and Eno Thereska for sharing the MSR Cambridge data center traces.

This work is dedicated to my dear mother and father.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

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, theenergy 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 [26]. Power consumption at such a level is almost enough to power all of Spain [1]. Energy-related costs are approaching the cost of IT hardware in data centers [7], and are growing 12% annually [48].

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., [32, 49, 43, 47] 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.

[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.

There have been substantial efforts in improving PUE, e.g., by optimizing cooling [45, 46] and power management [44]. In this thesis, we focus on reducing the energy consumed by the servers.

Real-world statistics reveal three observations that suggest that ample savings are possible in server energy consumption [11, 40, 12, 27, 15, 8]. 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 [36].

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 techniques, initiated by the pioneering work [11, 40] a decade ago. Among them, one line of work [36, 27, 12] 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 [11, 42, 12] 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 pieces of work address a number of schemes that deliver favorable performance justified by theoretic analysis and/or practical evaluations. See [4] 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 [12]. 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 performance of an online 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 offline solution that is computed with perfect future knowledge. Competitive analysis, adopting a worst-case mind set, allows us to access the robust performance guarantee for an online algorithm for arbitrary inputs ~~and arbitrary parameter settings~~. Recently, Lin *et al.* [31] 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 offline minimum. In simulations, they further show the algorithm can exploit available future workload information

[2]LCP algorithm [31] is a discrete time algorithm that only requires an estimate of the job arrival rate of the current slot.

to improve the performance. These results are very encouraging, indicating that a complete answer to the questions is possible.

1.2 Contributions

In this thesis, 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[3]. 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 $e/\left(e-1+\alpha\right)$. We prove that $2-\alpha$ and $e/\left(e-1+\alpha\right)$ 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

[3]The reason for this model is that the energy cost for typical servers is usually modeled as a linear function of the load [2]. However, the energy cost of current servers is dominated by the constant part cost [8]. Moreover, we will discuss later that our results can actually be applied to the case that idle and busy servers have different unit-time energy consumption.

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 believe utilizing future input information is a new and important design degree freedom for online algorithm.

- 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.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 introduces background and related work. We formulate the problem in Chapter 3. Chapter 4 reveals an important structure of the formulated problem, characterizes the optimal solution, and designs a simple decentralized offline algorithm achieving the optimal. In Chapter 5, we propose two online algorithms and provide performance guarantees. Chapter 6 presents the numerical experiments. Chapter 7 introduces a new angle for designing online algorithm and Chapter 8 concludes the thesis.

□ **End of chapter.**

Chapter 2

Related Work

A significant amount of work has focused on the issue of energy savings for a single processor, for an individual data center, and for multiple geographically separated data centers.

Theoretical study of energy use in a single speed-scaling processor starts from [51]. Yao et al. [51] proposed an offline algorithm that finds a minimum-energy schedule for any set of jobs under the assumption that the unit time power consumption P is a convex function of processor speed s. They also study two simple online heuristics: Optimal Available and Average Rate, and showed Average Rate has a constant competitive ratio for the case $P(s) = s^{\alpha}, \alpha \geq 2$. Paper [6] considers online dynamic frequency scaling algorithms to minimize the energy used by a server subject to the constraint that every job finishes by its deadline. The authors assume that the power required to run at frequency f is $P(f) = f^{\alpha}$ and show Optimal Available has a competitive ratio upper bounded by α^{α}. Paper [6] also proposes new online algorithm with competitive ratio $2\left[\alpha/\left(\alpha-1\right)\right]^{\alpha} e^{\alpha}$. The new algorithm is better then Optimal Available for large α in the light of competitive ratio. Stochastic analysis has also been applied to the problem of minimizing power consumption through speed scaling [50]. In paper [50], the authors try to optimally scale speed to balance mean response time and mean energy consumption under processor sharing scheduling. In the work [41], the authors

try to minimize the average response time of jobs, i.e., the time between their arrival and their completion of service, given the energy budget constraints.

For data centers with homogeneous servers, paper [18] tries to minimize the Energy-Response time Product (ERP) metric for a data center with single application servers which can only run on one frequency but can transition to many *sleep* states while server is idle. For a stationary demand pattern, [18] proves that there exists a very small, natural class of policies that always contains the optimal policy for a single server. For time-varying demand patterns, [18] proposes a simple, traffic-oblivious auto-scaling policy, DELAYEDOFF, to minimizing ERP, and the paper proves that as the average workload ρ goes to infinity, DELAYEDOFF will achieve optimal ERP asymptotically if server can transition from the *off* state to the *on* state instantly. In paper [31], the author proposed both offline and online algorithms to minimize the cost of a data center with single application environment. The paper proved that the online algorithm is 3-competitive, i.e., its cost is at most 3 times of that of the optimal offline solution. [42, 39] study the problem of minimizing the power cost for a data center by combining virtualization mechanisms and DVFS(Voltage and frequency scaling). They both propose optimization model and do simulation to evaluate their own model. For data centers with heterogeneous servers, work [11] designed an architecture to manage resource for internet hosting centers through adaptive resource provisioning. The main objective of resource management is to make the hosting centers more energy-efficient. [30, 38, 20] try to cut down the energy use for heterogeneous data center as well.

There also exists work aiming to improve overall energy-efficiency for multiple geographically separated data centers [30, 32, 29]. [29] tries to minimize the total energy cost for multiple internet data centers with location diversity and time diversity of electricity

price. They proposed a solution to the constrained mixed-integer programming problem they studied in the paper. They did experiments using real data showing that energy cost can be greatly reduced. [30] studies online algorithm "receding horizon control" (RHC) for geographical load balancing and shows that RHC performs well for homogeneous servers. They also provide variants of RHC with performance guarantee in the face of heterogeneity.

In the area of online algorithm designing, there is work trying to utilize some information of input to achieve better competitive ratio [17, 33, 21]. Paper [17] assumes that the input (time of skiing) of ski-rental problem is exponentially distributed. Under this assumption, they studied their problem using average-case competitive analysis and proposed optimal online strategy. In paper [33], the authors use semi-stochastic model rather than a fully stochastic model to handle input uncertainty in online optimization problems. Specifically, they explore the upper and lower bounds on the amount of stochastic information(online algorithm asks queries to obtain stochastic information about input and each query is confined to following queries: the algorithm gives a value $0 < s < 1$, and then the input gives a value l such that $\int_0^l p(t)\,dt = s$, where $p(x)$ is the real probability distribution of input. More stochastic information means more queries) required by a deterministic algorithm for the ski-rental problem to achieve a desired competitive ratio. In paper [21], the authors study online TSP(Traveling Salesman Problem) and TRP(Traveling Repairman Problem). They propose online algorithms which can utilize future information(they call advanced information) to improve competitive ratios achieved by previous work for the two problems. In paper [22], the authors propose both online deterministic and randomized algorithms for their server allocation problem with future information(no algorithms have bounded competitive ratios for this problem without future information). The paper proves that the ratio of the randomized algorithm is tight and the one

for deterministic algorithm is almost tight. The paper also shows that their approach can be used in a more general benefit task system. However, their approach works mainly with maximization problem not minimization problem.

In this thesis, the objective is to save energy cost for a data center. The problem studied in this thesis is similar to that studied in [31]. The difference is that we optimize a linear cost function over integer variables, while Lin et al. in [31] minimize a convex cost function over continuous variables (by relaxing the integer constraints). This thesis and [31] obtain different online algorithms with different competitive ratios for the two different formulations, respectively. For our formulation, we show that the competitive ratios of our algorithms can be significantly improved by exploiting the look-ahead information. We believe that looking-ahead provides a valuable degree of freedom in designing "future-aware" online algorithms with desirable competitive ratios. Comparing to [17, 33], we utilize future workload information in a look-ahead window in our online algorithms, and we study the cases where the future information in the window can be predicted accurately (in both analysis and experiments), or with prediction error (in experiments). Our setting is motivated by the observation that future workload information to some extent can be accurately predicted in data center [12, 10]. This is also the reason that we do not assume that future workload follows some (partially) known probability distribution in this thesis.

□ **End of chapter.**

Chapter 3

Problem Formulation

3.1 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[1], 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). [42, 13]. It is comparable to the energy cost of running a server for several hours [31].

The results we develop in this thesis 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 work [11, 40, 31, 14] 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. We assume that

[1] In practice, server's service capacity can be determined from the knee of its throughput and response-time curve [27].

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[2]. 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,

[2]This could be justified if there were a SLA in cloud computing that requires the job to 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. And the corresponding worst case is that at first there are m^2 jobs in the system and data center has to use at least m servers to process it. However, in the m^2 jobs there are only m jobs will last very long time and others will depart the system after a short time. In worst case, the algorithm happens to assign the m jobs to m different servers other than assign the m jobs to one server which is optimal offline dicision.

we further assume that a changes by at most 1 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

$$c_s(t_1, t_2) = P \int_{t_1}^{t_2} x_s(t)\, dt + \beta_{on}(u_s(t_2) - u_s(t_1)) + \beta_{off}(d_s(t_2) - d_s(t_1)) \tag{3.1}$$

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.

3.2 Problem Formulation

We formulate the problem of minimizing server operation cost in a data center in and 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:

$$\begin{aligned} & \mathcal{P}[a(t), X_1, X_2, T_1, T_2]: \\ \min \quad & P\int_{T_1}^{T_2} x(t)\,dt + P_{on}(T_1, T_2) + P_{off}(T_1, T_2) && (3.2) \\ \text{s.t.} \quad & x(t) \geq a(t), \forall t \in [T_1, T_2], && (3.3) \\ & x(T_1) = X_1, x(T_2) = X_2, && (3.4) \\ \text{var} \quad & x(t) \in \mathbb{Z}^+, t \in [T_1, T_2], && (3.5) \end{aligned}$$

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 power consumption 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$\int_0^T P_b a(t) - Pa(t)\,dt$ is constant for given $a(t)$, and so to minimize the total power consumption is to minimize (3.2). Constraints in (3.3) say the service capacity must satisfy the demand. Constraints in (3.4) 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 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 [31], 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 and left continuous when $a(t)$ increases one and decreases one, respectively. We remark that this simple modification will not change the optimal value of SCP.

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

□ **End of chapter.**

Chapter 4

Optimal Solution and Offline Algorithm

We study the offline 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.

4.1 Structure of Optimal Solution

We first define a critical interval as follows:

$$\Delta \triangleq \frac{\beta_{on} + \beta_{off}}{P} \tag{4.1}$$

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

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

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

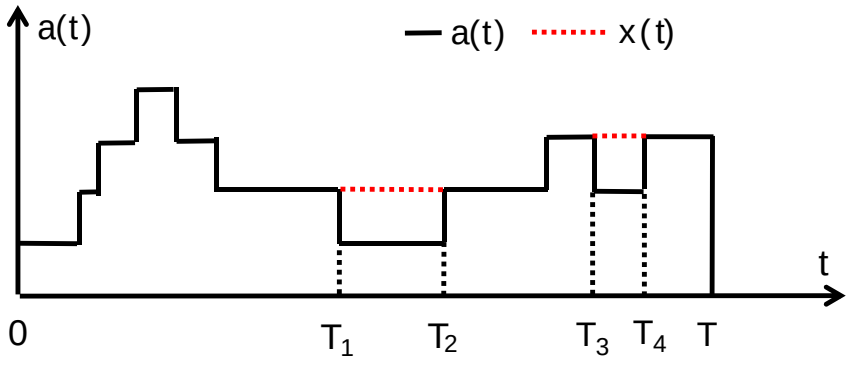


Figure 4.1: Example of solution constructed by Optimal Solution Construction Procedure.

Optimal Solution Construction Procedure:

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

Find all the intervals $(\tau, \tau^{'})$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 $(\tau, \tau^{'})$ **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. 4.1.

The following theorem is proved in Section A.1 using proof-by-contradiction and counting arguments.

Theorem 4.1.1. *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*

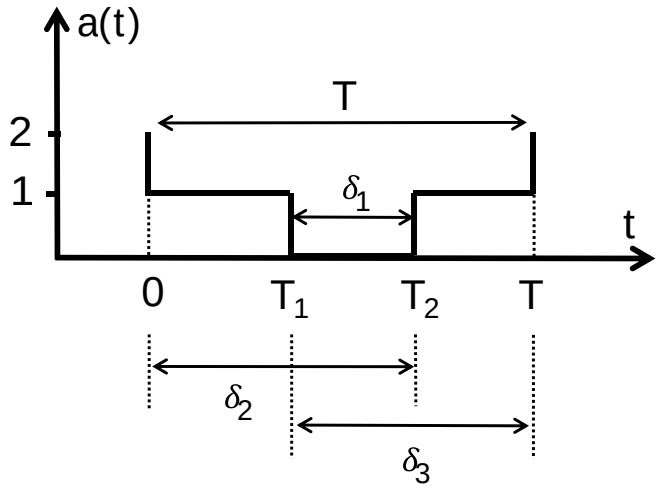


Figure 4.2: An example of a time period $[0, T]$. Interval $\delta_1 = T_2 - T_1$, $\delta_2 = T_2$, and $\delta_3 = T - T_1$.

limits.

4.2 Intuitions and Observations

Consider the example shown in Fig. 4.2. 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_1 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, based on which we build a decentralized offline 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

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 [35]. 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.

4.3 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 3.1, 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 offline algorithm that gives an optimal $x^*(t)$ and *incurs no job migration cost*.

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

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

By each server:

- Upon receiving a job: the server starts serving the job immediately.
- Upon a job leaving this server and it becomes empty: let the departure epoch be t_1. The server searches for the earliest time $t_2 \in (t_1, t_1 + \Delta]$ so that $a(t_2) = a(t_1)$. If no such t_2 exists, 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 offline algorithm.

Theorem 4.3.1. *The proposed offline algorithm achieves the optimal server operation cost of the problem SCP.*

Refer to Section A.2.

There are two important observations. First, the job-dispatching strategy only depends on the past job arrivals and departures. Consequently, the strategy assigns a job to the same server no matter it knows future job arrival/departure or not; it also acts independently to servers' off-or-idle decisions. Second, each individual server is actually solving a classic ski-rental problem [25] – whether to "rent", i.e., keep idle, or to "buy", i.e., turn off now and on later, but with their "days-of-skiing" (corresponding to servers' empty periods) *jointly determined by the job-dispatching strategy.*

Next, we exploit these two observations to extend the offline algorithm to its online versions with performance guarantee.

□ **End of chapter.**

Chapter 5

Online Dynamic Provisioning

Inspired by our offline algorithm, we construct online algorithms by combining (i) the same last-empty-server-first job-dispatching strategy as the one in the proposed offline 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 offline algorithm. This notion of comparison is called competitive analysis. 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 [1] if for all input sequences σ, we have

$$\mathbb{E}\left[\bar{C}_A(\sigma)\right] \leq RC_{opt}(\sigma) + O(1)$$

where $\mathbb{E}\left[\bar{C}_A(\sigma)\right]$ 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.

As discussed at the end of last section, the last-empty-server-first job-dispatching strategy utilizes only past job arrival/departure

[1] against an "oblivious adversary"

information. Consequently, as compared to the offline case, in the online case it assigns the same set of jobs to the same server at the same sequence of epochs. The following lemma rigorously confirms this observation.

Lemma 5.0.2. *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 offline and online situations.*

Lemma 5.0.2 is true because last-empty-server-first job-dispatching strategy only depends on past workload and it is independent on the historical statuses of servers. Hence, 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 offline and online situations.

As a result, *in the online case, each server still faces the same set of off-or-idle problems* as compared to the offline case. This is the key to derive the competitive ratios of our to-be-presented online algorithms.

Each server, not knowing the empty periods ahead of time, however, needs to decide whether to stay idle or be off (and if so when) in an online fashion. One natural approach is to adopt classic algorithms for the online ski-rental problem.

5.1 Dynamic Provisioning without Future Workload Information

For the online ski-rental problem, the break-even algorithm in [25] and the randomized algorithm in [24] 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 [31].

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 Δ amount of time. 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 5.2.2, it turns out this directly gives a 2-competitive dynamic provisioning solution.

5.2 Dynamic Provisioning with Future Workload Information

Classic online problem studies usually assume zero future information. However, in our data center dynamic provisioning problem, one key observation many existing solutions exploited is that the workload expressed highly regular patterns. Thus the workload information in a near look-ahead window may be accurately estimated by machine learning or model fitting based on historical data [12, 10]. Can we exploit such future knowledge, if available, in designing online algorithms? If so, how much gain can we get?

Let's 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 just becomes empty at time t_1, and its empty period happens to be just a bit longer than Δ.

Following the standard break-even algorithm, the server waits for Δ amount of time before turning itself off. According to the setting, it receives a job right 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.

An alternative strategy that costs less is as follows. The server stays idle for $(1-\alpha)\,\Delta$ amount of time, 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. According to the setting, 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 $(2-\alpha)\,P\Delta$ and is better than that of the break-even algorithm.

This simple 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 $a(t)$ for 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 resulting 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:
By a central job-dispatching entity: it implements the last-empty-server-first job-dispatching strategy, i.e., the one described in the offline algorithm.
By 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 for $(1-\alpha)\,\Delta$ amount of time,
  - 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 5.2.2 later in this section, the algorithm CSR has the best possible competitive ratio for any deterministic algorithms under the last-empty-server-first job-dispatching strategy. Thus, unless we change the job-dispatching strategy, 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 [24], but extended to consider future information. However, if servers turn off after random times, then it is possible that the last-empty server is off even though there are idle servers that are on. Instead of using the last-empty server, we will dispatch jobs to the server, if any, that is on but has been idle least time, as done in [18].

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:
By a central job-dispatching entity: it implements the least-idle job-dispatching strategy.
By each server:

- Upon receiving a job: reset all timers, and start serving the job immediately.

- Upon a job leaving this server, record the occupancy as a, and initialize a timer to expire time Z into the future, where Z is distributed as

$$f_Z(z) = \frac{\exp(\{z/(1-\alpha)\Delta\})}{(e-1+\alpha)(1-\alpha)\Delta}\mathbf{1}_{0<z\leq(1-\alpha)\Delta} + \frac{\alpha}{e-1+\alpha}\delta(z) \tag{5.1}$$

  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 Section A.3, shows that RCSR performs at least as well as it adopting last-empty-server-first job-dispatching strategy, which will allow us to obtain a competitive ratio.

Lemma 5.2.1. *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 offline 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. The algorithms are decentralized (except for the prediction mechanism), making them easy to implement and scale.

Observing no such future-aware online algorithms available in the literature, we analyze their competitive ratios and present

the results as follows. Assume that jobs assigned to a server is countable.

Theorem 5.2.2. *For any $P, \beta_{off}, \beta_{on}$, the online algorithms CSR and RCSR have competitive ratio of $2-\alpha$ and $e/(e-1+\alpha)$. The competitive ratios of CSR and RCSR are the best possible for deterministic and randomized algorithms, respectively, under the last-empty-server-first job-dispatching strategy.*

Refer to Section A.4.

Remarks: (i) When $\alpha = 1$, all two algorithms achieve the optimal server operation cost. This matches the intuition 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 first critical interval Δ (corresponding to $\alpha = 1$) will not improve dynamic provisioning performance. (ii) The competitive ratios presented in the above theorem is 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. 5.1. (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 6. (v) In the thesis, we propose CSR/RCSR for data centers with homogeneous servers. In fact, the proposed algorithms can also be applied to date centers with following heterogeneous servers for mice workload(CSR and RCSR can be applied to this kind of workload with few modifications. Detailed discussion of those modifications in Section 5.3): servers have the same one-time switching cost but

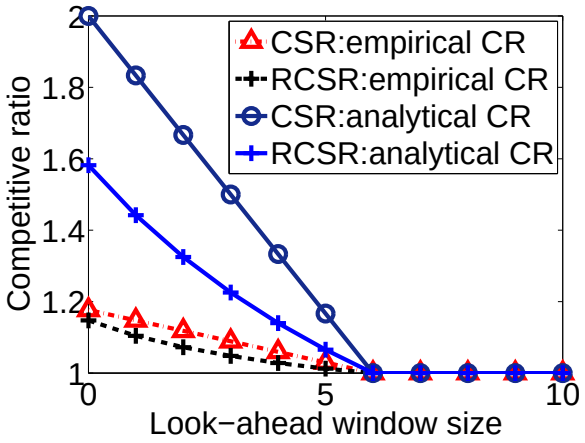


Figure 5.1: Comparison of the worst-case competitive ratios (according to Theorem 5.2.2) and the empirical competitive ratios observed in simulations using real-world traces. The full-size look-ahead window size $\Delta = 6$ units of time. More simulation details are in Chapter 6.

different unit time power consumption. In this heterogeneous scenario, CSR and RCSR will use servers with lower unit time power consumption first in each slot. And each server solves its own ski-rental problem independently. It can be verified that CSR and RCSR still have competitive ratios of $2 - \alpha$ and $e/(e - 1 + \alpha)$ in this case. Extending CSR and RCSR to general heterogeneous cases is a future direction for this thesis.

In our algorithms, we assign the servers which were most recently busy to new coming jobs. An alternative approach is that servers which haven't served jobs for the longest time are dispatched to new jobs(we call this as longest-waiting-server-first strategy), and individual server solves its ski-rental problem independently. This new approach may seem fairer than our algorithms. However, it is not energy efficient both from the perspective of competitive analysis and in our case study.

For the competitive analysis, let workload be the curve shown in Fig. 5.2. The service time for each job is exactly $\delta/2$ units of time and a new job is coming $\delta/2$ units of time after the departure of the previous job. And the number of active servers(idle or busy

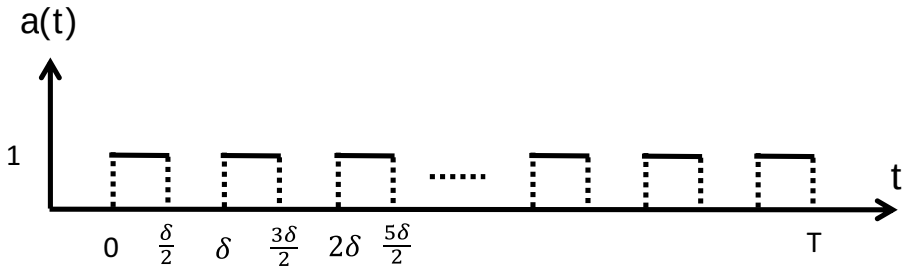


Figure 5.2: One example of workload for which the ratio of the cost of the alternative longest-waiting-server-first strategy to the offline optimal can be arbitrarily bad.

servers) in data center at time 0 is N. δ is chosen such that $N\delta = \Delta$. In this scenario, the offline optimal policy would turn off $N-1$ servers at the beginning and leave one server on and run this server to serve all the jobs. The optimal energy consumption is $PT + N\beta_{on} + N\beta_{off}$. For this workload, CSR will turn off $N-1$ servers at time Δ and just run one server to serve all the jobs because CSR will assign the server which has waited for the least time to new job. Therefore the total cost of CSR is $PT + (N-1)P\Delta + N\beta_{on} + N\beta_{off}$(RCSR is similar to CSR but randomly choose a time to turn off $N-1$ servers, therefore the cost of RCSR is less than that of CSR).

Meanwhile, due to the longest-waiting-server-first strategy, in the new approach these N servers will be running during the whole period. This is because there are exactly N jobs coming into the data center during the waiting period of Δ units of time after a particular server becoming empty. And when the Nth job arrives this particular server would be assigned to serve the job because it is the one which has not served a job for the longest time. Thus the total energy cost for this approach is $[NPT + N\beta_{on} + N\beta_{off}]$. Since N and PT can be much larger than $\beta_{on} + \beta_{off}$, the ratio of total energy cost of the new approach to the offline optimal is close to N in this case. Another important observation is that as compared to the alternative approach, CSR behaves more like

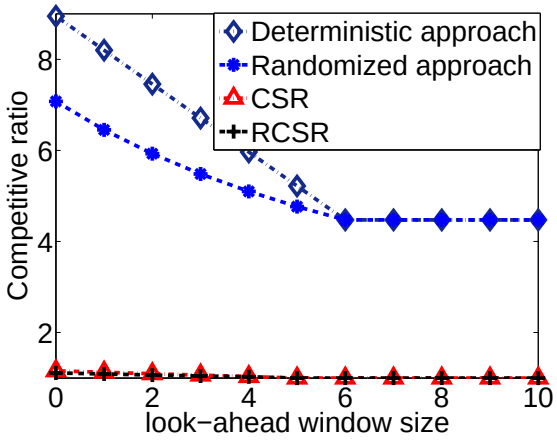


Figure 5.3: Comparison of the empirical competitive ratios of the alternative approach with longest-waiting-server-first, its randomized version, CSR and RCSR. The full-size look-ahead window size $\Delta = 6$ units of time.

the offline optimal policy(just use one server to serve all the jobs) and cost much less energy. This demonstrates that least-waiting-server-first(equivalent to last-empty-server-first in CSR and least-idle in RCSR) is better than longest-waiting-server-first strategy. We can get similar conclusion for its randomized version(with the same job-dispatching strategy but each server adopts the best randomized ski-rental algorithm [24]).

In our case study, we evaluate this alternative approach and its randomized version using real-world trace and compare their performance to that of CSR/RCSR. The results are shown in Fig. 5.3, which indicate that the empirical competitive ratios of these alternative approaches are much worse than those of CSR/RCSR.

Note that our algorithms are closely related to the DELAYED-OFF algorithm in [18], 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 [18]). 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 [18] considers expected performance in a stochastic setting and a large-system asymptotic regime.

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

Adapting our offline 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 chopped 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 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.

For individual servers, they start to serve upon receiving jobs, and start to solve the offline or online ski-rental problems upon all its jobs leaving and it becomes empty. It is not difficult to verify the modified algorithms still retain their corresponding performance guarantees. Actually, we have following corollary.

Corollary 5.3.1. *The modified deterministic and randomized online algorithm for discrete-time fluid workload have competitive ratios of* $2-\alpha$ *and* $e/(e-1+\alpha)$*, respectively.*

5.4 Extending to Case Where Servers Have Setup Time.

Until now, we have ignored the time T_s required for a server to turn on. We will extend our algorithms CSR and RCSR to the case where servers take T_s to turn on in this section. We now describe a centralized algorithm EXT that provides a bounded CR in the case where $a(\tau) \leq (1+\gamma)a(t)$ for all $\tau \in [t, t+T_s]$. This is to say workload increases at most by a factor of $(1+\gamma)$ in any interval of length T_s. Define $a_{\min} = \min_{t\in[0,T]} a(t)$. Our "bounded-increment" model imposes a requirement that $a_{\min}$ has to be no less than $1/\gamma$. This is because the number of job increases at least by 1 and we must therefore have $\gamma a_{\min} \geq 1$, which indicates that $a_{\min} \geq 1/\gamma$. The requirement that $a_{\min}$ needs to be larger than $1/\gamma$ is not difficult to satisfy in practice and hence it does not limit the practical relevance and applicability of our model and the following analysis. For instance, in [34] it suggests a typical value of γ is around 10%. This in terms requires $a_{\min}$ to be large enough to be served by at least 10 servers, for our model and analysis to be applicable. But such requirement is easily satisfied in any realistic data centers or server racks such as the Akamai ones [34].

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 the last-empty-first strategy in CSR or the least-idle-first strategy in RCSR. 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 A.5, establishes the validity and performance guarantees of EXT.

Corollary 5.4.1. *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* $a(t)$ *. 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) Note that $1/\gamma$ is a lower bound for $a_{\min}$, following which we can get the stated upper bounds of the competitive ratios of EXT. (ii) The upper bound of competitive ratio of EXT is linearly proportional to γ. (iii) Since the minimal workload $a_{\min}$ in large data centers is usually much larger than that in small ones, EXT is more beneficial for large data center because the competitive ratio is smaller. (iv) In EXT, we adopt over-provisioning to combat the problem that servers need setup time T_s; 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 when servers have a setup time constraint.

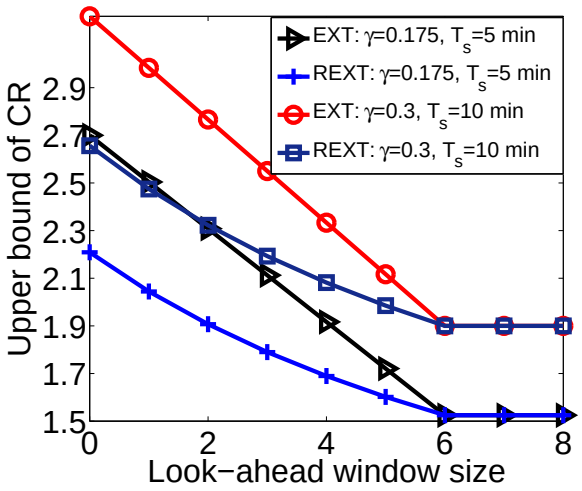


Figure 5.4: Upper bound of the worst-case competitive ratios (according to Corollary 5.4.1). The full-size look-ahead window size is 1 hour.

Fig. 5.4 shows the relationship between the upper bounds of competitive ratios of EXT and the setup time T_s for a workload trace which increases at most by a factor of 1.3 in any interval with length 10 minutes and at most by a factor of 1.175 in any interval with length 5 minutes(More details about the workload and other simulations in Chapter 6). As indicated in Fig. 5.4, the shorter the setup time T_s, the better the competitive ratio of EXT. This is because workload increases less dramatically when setup time T_s is shorter, hence the over-provisioning of EXT is less than that of longer setup time.

□ **End of chapter.**

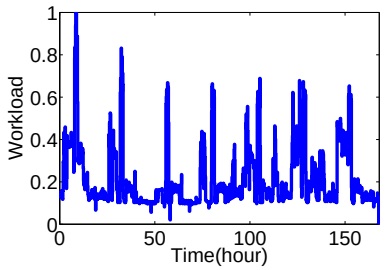
Chapter 6

Experiments

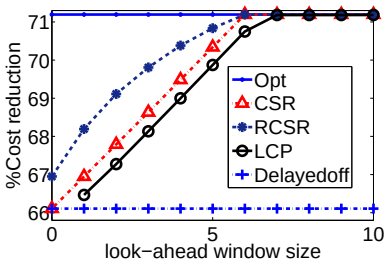
We implement the proposed offline and online algorithms and carry out simulations using real-world traces to evaluate their performance. Our aims are threefold. First, to evaluate the performance of the algorithms in a typical setting. Second, to study the impacts of workload prediction error and workload characteristics on the algorithms' performance. Third, to compare our algorithms to two recently proposed solutions LCP(w) in [31] and DELAYEDOFF in [18].

6.1 Settings

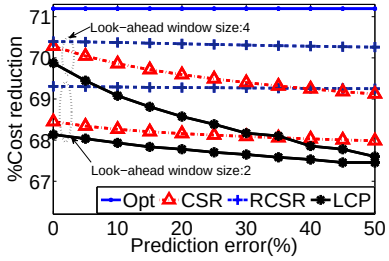
Workload trace: The real-world traces we use in experiments are a set of I/O traces taken from 6 RAID volumes at MSR Cambridge [37]. The traced period was one week from February 22 to 29, 2007. We estimate the average number of jobs over disjoint 10 minute intervals. The data trace has a peak-to-mean 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.



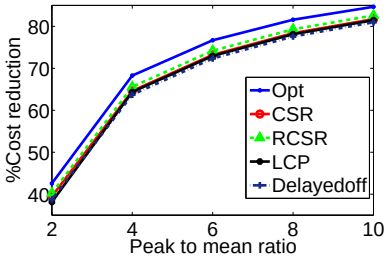
(a) Trace for one week in MSR



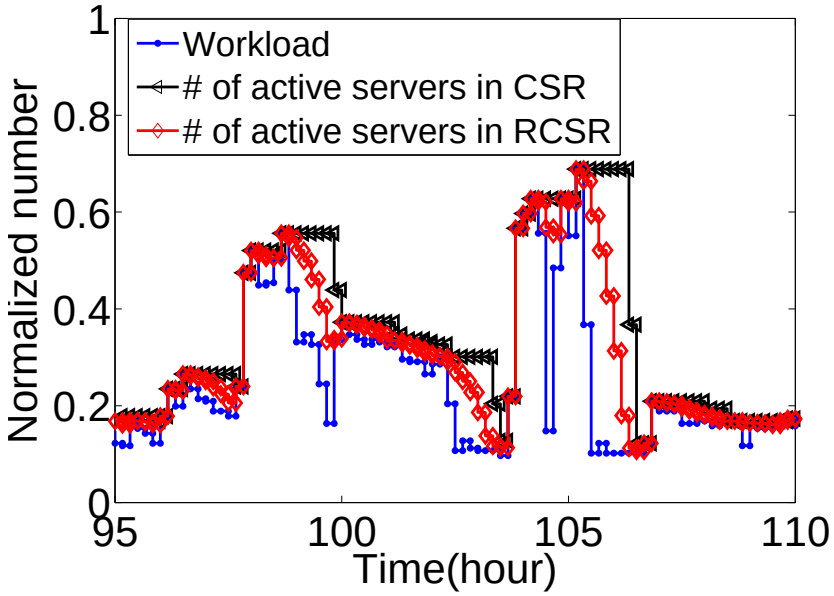
(b) Impact of future information



(c) Impact of prediction error



(d) Impact of PMR



(e) Normalized numbers of jobs and active servers given by CSR/RCSR in a period

Figure 6.1: Real-world workload trace, normalized number of servers given by CSR/RCSR 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 6.5.

In the experiments, we run algorithm LCP(w) [31] 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 [18] 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 job-dispatching strategy 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 [31, 27]. 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 [31]. Under this setting, the critical interval is $\Delta = (\beta_{off} + \beta_{on}) / P = 6$ units of time.

6.2 Performance of the Proposed Online Algorithms

We have characterized in Theorem 5.2.2 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 scenarios. 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 cost reduction are shown in Fig. 6.1b. The cost reduction curves are obtained by comparing the power cost incurred by the offline algorithm, CSR, RCSR, the LCP(w) algorithm [31] and the DELAYEDOFF algorithm [18] 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.

The curves of normalized numbers of servers(normalized by the maximal number of servers in $[0,T]$) given by CSR/RCSR are shown in Fig. 6.1e. In order to show more details, only the numbers of active servers in a period(from hour 95 to hour 110) are plotted. Fig. 6.1e indicates that the numbers of active servers used by CSR and RCSR decrease when workload is low and increase when workload is high and similar patterns are observed in the rest period. Those curves actually match the intuitive strategy that in order to save energy we should turn on just enough servers to meet the demand. It can also be seen that RCSR is more aggressive in turning servers off as compared to CSR, which for this workload trace leads to the observation that RCSR reduces more operating cost than CSR.

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 5.2.2 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.

6.3 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 [28]; thus we are essentially stress-testing the algorithms.

We average 100 runs for each algorithm and show the results in Fig. 6.1c, where the vertical axis represents the cost reduction as compared to the benchmark.

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.

6.4 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 work [31, 27]. Similar to [31], 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 offline 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. 6.1d.

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.

6.5 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. 6.1b 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. 6.1b, 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 [31], 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.

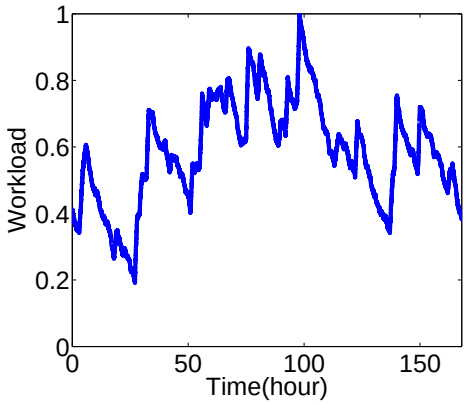
6.6 Additional Experiments

In this section, we will evaluate CSR and RCSR with "elephant" workload. Since we do not have any real data center trace of this kind of workload, we generate a synthetic workload as shown in Fig. 6.2a. In this "elephant" workload trace, the job arrival is Poisson process and the arrival rate is constant within an hour but varies across the intervals of length one hour. The service time of each job is exponentially distributed and the mean is about 33 hour. The PMR of this trace is 1.7. The simulation result is shown in Fig. 6.2b. One observation from Fig. 6.2b is that the CSR and RCSR energy-saving curves of the "elephant" workload are similar to that of the "mice" workload as shown in Fig. 6.1b. Our algorithms can save more than 37% energy. This number is consistent with the result in Fig. 6.1d, which suggests that the

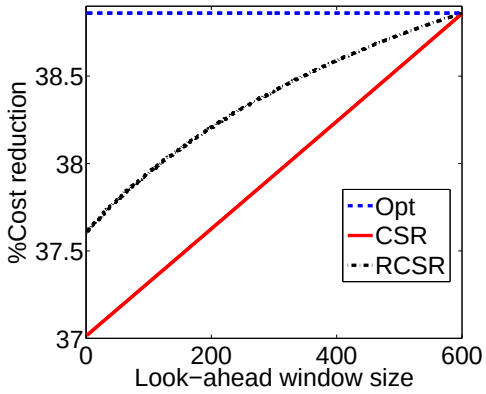
energy saving is about 40% when PMR is 2.

The workload increases at most by a factor 1.3 in 10 minutes and by a factor 1.175 in 5 minutes. We evaluated our algorithm EXT. Fig. 6.2c shows that our algorithm EXT can save more than 20% energy when comparing to static provisioning. Moreover, the empirical competitive ratio of is much smaller then the analytic one in Fig. 6.2d (EXT is the algorithm extended from CSR; REXT is the one extended from RCSR). This is because i) the analytic competitive ratio for online algorithm is for worst case and the particular trace we used here may not be the worst case; ii) we use the upper bound of competitive ratios as the theoretical results.

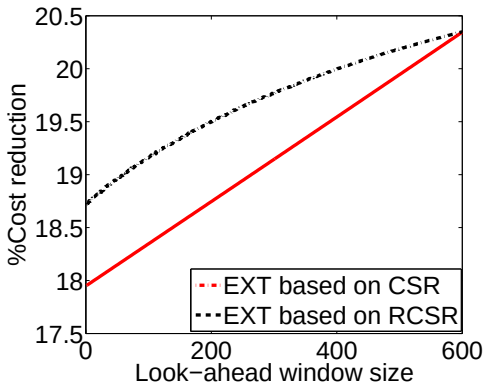
□ **End of chapter.**



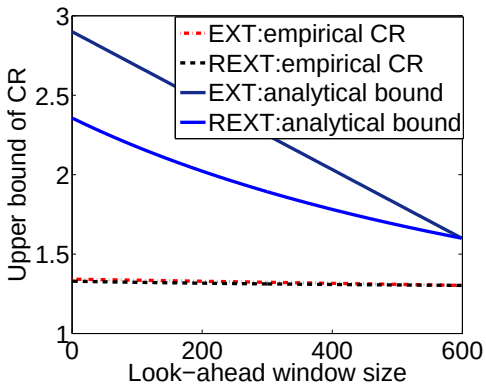
(a) Synthetic "Elephant" Workload.



(b) Energy saving of our algorithms for "elephant" workload.



(c) Energy saving of EXT.



(d) Competitive ratios of EXT.

Figure 6.2: Experiments

Chapter 7

A New Degree of Freedom for Designing Online Algorithm

In the preceding chapter, we proposed algorithms CSR and RCSR which can utilize future information to achieve better performance. This suggests a new angle for designing online algorithm: utilizing future information of input to get better performance. In online algorithm design, most of the existing work focus on the worst case analysis in which online algorithms do not know any information of the future input and the performance of the online algorithm is determined by its performance under its worst case (such as competitive ratios). There is also work that evaluates online algorithms using average case analysis. In this case, online algorithms know the probability distribution of the input and the performance of the online algorithms is decided by its average performance over all possible inputs [17]. In this chapter, we will first introduce a classic online problem named the lost cow problem and its online algorithms which are analyzed with in the worst case. Then we will show how to utilize future input information to improve the performance of the online algorithms. We will also take secretary problem and its online algorithm with average case analysis as an example to illustrate how online algorithm can benefit from future input information.

7.1 The Lost Cow Problem

Image a totally short-sighted cow (can only see things right before it) is at point S on a line and it wants to find a target at point T on the same line. However, the cow dose not know whether the target is located to its right or left, neither does it know the distance $d \geq 1$ between S and T. The objective of the cow is to minimize the total distance it will walk in order to find the target at T.

This problem was introduced by [5] which also proposed a "doubling" strategy to achieve the optimal competitive ratio of 9 for deterministic algorithm. The algorithm performs as follows:

1. The cow first walks to the right for a distance of 1, it will go back to S if it fails to reach the target;
2. After a failed search, the cow will go in the opposite direction for a distance which is double of that of the previous search. If the cow fails to reach the target, it will go back to S;
3. Repeat step 2 until the cow reaches the target at T.

The above algorithm achieves the best competitive ratio of 9 for deterministic algorithm. In paper [23], the author proposed a randomized algorithm, called SmartCow, achieving the optimal competitive ratio for randomized algorithms. Define $r = \operatorname{argmin}_{x>1} \frac{1+x}{\ln x} \approx 3.59$, the randomized algorithm is as follows:

1. The cow first randomly and uniformly choose a direction and generate a number ε which is uniformly distributed $[0, 1)$, then walks in the chosen direction for a distance of r^{ε}, it will go back to S if it fails to reach the target;
2. After a failed search, the cow will go in the opposite direction for a distance which is r times of that of the previous search. If the cow fails to reach the target, it will go back to S;

3. Repeat step 2 until the cow reaches the target at T.

This randomized algorithm has competitive ratio $1 + \frac{1+r}{\ln r} \approx 4.59$ [23].

It turns out that with future input information, we can do better for the online lost cow problem. Assume that the cow is not totally short-sighted and can see thing clearly within a look-ahead window of length of $l = \theta d$, where $0 \leq \theta \leq 1$. The cow can follow the following deterministic and randomized algorithms:

1. If the cow can see the target at T at S, it will go directly to the target; Otherwise, the cow first walks to the right for a distance of 1. If it can not see the target at T in the look-ahead window, it will go back to S; Otherwise, it will go directly to the target at T.

2. After a failed search, the cow will go in the opposite direction for a distance which is double of that of the previous search. If the cow fails to reach the target or see the target in the look-ahead window, it will go back to S; Otherwise, it will go directly to the target at T.

3. Repeat step ii) until reach target T.

SmartCow(θ) is the randomized algorithm.

1. If the cow can see the target at T at S, it will go directly to the target; Otherwise, the cow first randomly and uniformly choose a direction and generate a number ε which is uniformly distributed in $[0, 1)$, then walks in the chosen direction for a distance of r^{ε}, if it can not see the target at T in the look-ahead window, it will go back to S; Otherwise, it will go directly to the target at T.

2. After a failed search, the cow will go in the opposite direction for a distance which is r times of that of the previous search. If the cow fails to reach the target or see the target in the

look-ahead window, it will go back to S; Otherwise, it will go directly to the target at T.

3. Repeat step ii) until reach target T.

With future information, the cow can see the target even if it is not very close to the target. Therefore, the cow can travel less distance than that without future information because in the later case the cow will explore the opposite direction if the target is not close to itself. Define $\overline{\theta}$ such that $(1-\overline{\theta})d = 1$. In fact, we have following lemma:

Lemma 7.1.1. *If the cow can see things clearly within a distance of $l = \theta d$, where $0 \leq \theta \leq 1$, then above deterministic algorithm and randomized algorithm SmartCow(θ) have competitive ratios of i) $9 - 8\theta$ and $1 + \frac{1+r}{\ln r} - \frac{1+r}{\ln r}\theta \approx 4.59 - 3.59\theta$ when $\theta < \overline{\theta}$; ii) $3 - 2\overline{\theta}$ and $1 + \frac{r-1}{\ln r} - \frac{r-1}{\ln r}\overline{\theta} \approx 3 - 2\overline{\theta}$ when $\overline{\theta} \leq \theta < 1$; iii) 1 and 1 when $\theta = 1$, respectively.*

Refer to Section A.6.

7.2 Secretary Problem without Future Information

It turns out many online problems can utilize future information to have better competitive ratio. One example is the classic secretary problem [16].

Imagine an employer wants to hire a secretary from n applicants which the employer can rank from the best to the worst without ties. Moreover, the capability of an applicant can only be determined after interview and the employer interviews the candidates in a random order. After interviewing an applicant, the employer has to make a decision immediately and irrevocably whether to hire the applicant, or move on to the next one. The

objective of the employer is to maximize the probability of hiring the best applicant.

This problem is known as the secretary problem. An optimal strategy for the employer is to interview and reject the first k applicants, k is to be determined. After rejecting the first k applicants, the employer hires the first applicant who is better than all the applicants previously interviewed. If no such applicants exist, the employer hires no one. With this policy, according to [16] the probability of hiring the best applicant is

$$\sum_{j=k+1}^{n} \frac{1}{n} \frac{k}{j-1} \tag{7.1}$$

The optimal k for this strategy is the one maximizing the total probability (7.1). For small n, the optimal k can be easily computed. We are more interested in finding the approximate value of the optimal k for large n. For large n, we have $\sum_{j=k+1}^{n} \frac{1}{n} \frac{k}{j-1} \approx \frac{k}{n} \ln \frac{n}{k}$. Hence, it is easy to find the optimal $k = \frac{n}{e}$ and the employer has probability $1/e$ [16] to hire the best applicant.

7.3 Secretary Problem with Future Information

Without any future information, $1/e$ is the best we can do for large n [19]. However, the probability can be improved if future information of next $m = \theta n$ applicants is known when the employer makes a decision to hire or reject a specific applicant. In reality, knowing the future information can be seen as result of following policy: the employer first interviews $m + 1$ candidates, then decide whether to hire the one being earliest interviewed among the

$m+1$ candidates or not. If the earliest one is rejected, the employer interviews another applicant and then decides to hire the new "earliest" applicant or not. This process will be repeated until the employer reaches the last applicant of the total n candidates.

For secretary problem with future information, the online solution is similar to that without future information:

- The employer interviews and rejects the first k applicants, k is to be determined.
- After rejecting the first k applicants, the employer stops at an applicant who is better than all the applicants previously interviewed and hires the best applicant among the $m+1$ applicants (the one currently being interviewed and the m applicants that employer can foresee their capabilities without interview.)

The following theorem justify that the above online solution improve the probability of selecting the best one.

Theorem 7.3.1. *When n is large, let $k = (1-\theta)\, e^{-\frac{1}{1-\theta}}$ in above online solution, the probability of selecting the best applicant is $\theta + (1-\theta)\, e^{-\frac{1}{1-\theta}}$.*

Refer to Section A.7.

We remark that with future information we can improve the probability to hire the best candidate. This is possible because we can explore and reject less candidates (less then n/e candidates) but gain roughly the some amount of information of the capability of applicants as that without future information. Moreover, when we select one applicant we select the best one from $m+1$ candidates instead of just one in the case without future information. Therefore, the probability can be enhanced with future information.

7.4 Summary

Based on the three cases: ski-rental problem faced by a server in CSR and RCSR, the lost cow problem and the secretary problem, we believe utilizing future information of the input is a new and important design degree freedom for online algorithm. In traditional online algorithm design, future input information is usually not taken into account. Perhaps as a result of that, many online problems have simple online algorithms with optimal but large competitive ratios. Since future input information to some extent can be estimated accurately in many online problems, hence we believe future input information can be harnessed in online algorithm design to achieve better competitive ratio and provide more competitive edge in both practice and theory.

□ **End of chapter.**

Chapter 8

Conclusion

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

We reveal an elegant "divide-and-conquer" structure of the offline 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 $0 \leq \alpha \leq 1$ 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 $e/\left(e-1+\alpha\right)$. The ratios $2-\alpha$ and $e/\left(e-1+\alpha\right)$ 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 thesis is similar to that

studied in [31]. The difference is that we optimize a linear cost function over integer variables, while Lin et al. in [31] minimize a convex cost function over continuous variables (by relaxing the integer constraints). This thesis and [31] 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. We also believe utilizing future input information is a new and important design degree freedom for online algorithm.

In addition, we also propose online algorithms for the case that servers need setup time T_s but the load satisfies $a(\tau) \leq (1+\gamma)a(t)$ 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. When utilizing future workload information, we assume that the future information is accurate in the look-ahead window in our algorithm design and competitive analysis, and we study in experiments the performance of the proposed algorithms when the future information is not perfectly known. An interesting and important future direction is to design competitive online algorithms that can utilize inaccurate future input information. Such algorithms will be very attractive in practice, where prediction of the future input information often comes with errors.

Another important direction would be to extend these results to the general case of heterogeneous servers or multiple geographically separated data centers [30, 38, 20]. It would be useful to extend the insight from this thesis to heterogeneous cases.

□ **End of chapter.**

Appendix A

Proof

The claims made in the previous sections will now be proven.

A.1 Proof of Theorem 4.1.1

In order to prove theorem 4.1.1, we introduce three lemmas. The first establishes that P_{on} and P_{off} are well defined.

Lemma A.1.1. *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 A.1.2. *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 power consumption 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 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 $(T_e - T_s) > \triangle$, $x_i(t)$ actually cost more power 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 consume more on-off cost or operating cost than $x_i(t)$, which means $\overline{x}(t)$ is an optimal solution. □

Lemma A.1.3. *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 4. 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 A.1.2, 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 $\hat{x}(t)$ cause less power consumption than $\bar{x}^*(t)$ in $[0, T]$ and they have the same power consumption in the rest periods. It is a contraction that $\bar{x}^*(t)$ is an optimal solution. Therefore, we have that $\bar{x}^*(t), t \in [0, T]$ is an optimal solution to SCP. □

Next, we are going to prove Theorem 4.1.1.

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 (A.1.2), $\bar{x}^*(t) \leq a(\mu), \forall t \in (\tau, \tau')$ 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 (τ_1, τ_1') and (τ_2, τ_2') which have following properties:

(1) (τ_1, τ_1') covering (τ, τ'), $\bar{a}(\tau_1) \geq \bar{a}(\mu) + 1, \bar{a}(\tau_1') \geq \bar{a}(\mu) + 1$ and $\tau_1' - \tau_1 \leq \Delta$. Moreover, for any interval $(\upsilon_1, \upsilon_1^1)$ cov-

ering $\left(\tau_1, \tau_1^{'}\right)$ and $\bar{a}\left(\upsilon_1\right) \geq \min\left[\bar{a}\left(\tau_1\right), \bar{a}\left(\tau_1^{'}\right)\right] + 1$, $\bar{a}\left(\upsilon_1^{'}\right) \geq \min\left[\bar{a}\left(\tau_1\right), \bar{a}\left(\tau_1^{'}\right)\right] + 1$, we must have $\upsilon_1^{'} - \upsilon_1 > \Delta$.

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

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

On the other hand, according to lemma (A.1.2),
$\bar{x}^*\left(t\right) \leq \min\left[\bar{a}\left(\tau_1\right), \bar{a}\left(\tau_1^{'}\right)\right], \forall t \in \left(\tau_2, \tau_2^{'}\right)$ because $\bar{x}^*\left(t\right)$ is an optimal solution to $\mathcal{P}\left[\bar{a}\left(t\right), \bar{a}\left(-2\Delta\right), \bar{a}\left(T+2\Delta\right), -2\Delta, T+2\Delta\right]$. It is clear that $\bar{x}^*\left(t\right) \geq \min\left[\bar{a}\left(\tau_1\right), \bar{a}\left(\tau_1^{'}\right)\right], \forall t \in \left(\tau_1, \tau_1^{'}\right)$ because turning server off and on later cause no less power that just let server be idle during $\left(\tau_1, \tau_1^{'}\right)$ since $\tau_1^{'} - \tau_1 \leq \Delta$. Therefore, we must have $\bar{x}^*\left(\mu\right) = \min\left[\bar{a}\left(\tau_1\right), \bar{a}\left(\tau_1^{'}\right)\right]$. This means in Case II our Optimal Solution Construction Procedure also gives an optimal solution.

Thus the $x\left(t\right)$ constructed by the Optimal Solution Construction Procedure is an optimal solution to
$\mathcal{P}\left[\bar{a}\left(t\right), \bar{a}\left(-2\Delta\right), \bar{a}\left(T+2\Delta\right), -2\Delta, T+2\Delta\right]$. Due to lemma (A.1.3), we have $x\left(t\right), t \in [0, T]$ constructed by the Optimal Solution Construction Procedure is an optimal solution to SCP. □

A.2 Proof of Theorem 4.3.1

First we are going to prove that the offline can solve
$\mathcal{P}\left[\bar{a}\left(t\right), \bar{a}\left(-2\Delta\right), \bar{a}\left(T+2\Delta\right), -2\Delta, T+2\Delta\right]$ optimally. Due to lemma (A.1.3), the offline algorithm can also solve SCP problem optimally. First, we are going to prove following lemma.

Lemma A.2.1. *Under last-empty-server-first job-dispatching strategy, if a server becomes empty at τ_1 and it will receive the first job after τ_1 at $\tau_1^{'}$, then we have $\bar{a}\left(\tau_1\right) = \bar{a}\left(\tau_1^{'}\right)$ and $\bar{a}\left(t\right) < \bar{a}\left(\tau_1\right), \forall t \in \left(\tau_1, \tau_1^{'}\right)$.*

Proof. It is clear that 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 job-dispatching entity. Assume that the server's ID becoming empty at τ is $\mathcal{S}$. Since $\tau_1^{'}$ is the first time that $\mathcal{S}$ is poped out after τ, then the number of servers' ID on the top of $\mathcal{S}$ does not change during $\left(\tau_1, \tau_1^{'}\right)$. Therefore, we have $\bar{a}\left(\tau_1\right) = \bar{a}\left(\tau_1^{'}\right)$ and $\bar{a}\left(t\right) < \bar{a}\left(\tau_1\right), \forall t \in \left(\tau_1, \tau_1^{'}\right)$ because the number of jobs in the system is equal to the number of server's IDs that are not in the stack. □

Lemma A.2.2. *For any idle server at time μ, assume that the idle server become empty at τ. Then we have $\bar{a}\left(\tau\right) > \bar{a}\left(\mu\right)$.*

Proof. Assume that the idle server will receive its first job after μ at $\tau^{'}$, according to lemma A.2.1, we $\bar{a}\left(t\right) < \bar{a}\left(\tau_1\right), \forall t \in \left(\tau_1, \tau_1^{'}\right)$. Thus, we have $\bar{a}\left(\tau\right) > \bar{a}\left(\mu\right)$. □

Now, we are going to prove theorem 4.3.1. The proof is similar to the proof of theorem 4.1.1. Let $x_o\left(t\right)$ Denote the number of servers run by offline algorithm at t.

Proof. For any $\mu \in [0, T]$, we must have that μ is in some interval $\left(\tau, \tau^{'}\right)$ such that $\bar{a}\left(\tau\right) \geq \bar{a}\left(\mu\right) + 1$, $\bar{a}\left(\tau^{'}\right) \geq \bar{a}\left(\mu\right) + 1$ and $\bar{a}\left(t\right) \leq \bar{a}\left(\mu\right), \forall t \in \left(\tau, \tau^{'}\right)$. 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 $x\left(\mu\right) = \bar{a}\left(\mu\right) = a\left(\mu\right)$.

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) if $\tau = -2\Delta$.

In this sub-case, if there is idle servers at μ, then some idle servers will receive jobs at $\tau^{'}$. According to lemma A.2.1, there must exist a time υ in $\left[0, \tau^{'}\right)$ such that $\bar{a}\left(\nu\right) > \bar{a}\left(\mu\right)$, which contradicts that $\bar{a}\left(t\right) \leq \bar{a}\left(\mu\right), \forall t \in \left(-2\Delta, \tau^{'}\right)$.

(2) if $\tau^{'} = T + 2\Delta$.

In this sub-case, if there is idle servers at μ, according to lemma A.2.1 and A.2.2, 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) $\left(\tau, \tau^{'}\right) \in [0, T]$.

In this sub-case, if there is idle servers at μ, according to lemma A.2.1 and A.2.2, the idle server will receive a job after $\tau^{'}$. Thus the idle period for the idle server is larger than $\tau^{'} - \tau > \Delta$, which contradicts that in our offline algorithm the idle period for a server is at most 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 $\mathcal{P}\left[\bar{a}(t), \bar{a}(-2\Delta), \bar{a}(T + 2\Delta), -2\Delta, T + 2\Delta\right]$.

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

(2) $\left(\tau_2, \tau_2^{'}\right)$ covering $\left(\tau_1, \tau_1^{'}\right)$, $\bar{a}(\tau_2) \geq \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1^{'}\right)\right] + 1$, $\bar{a}\left(\tau_2^{'}\right) \geq \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1^{'}\right)\right] + 1$, $\tau_2^{'} - \tau_2 > \Delta$ 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 $x(\mu) = \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1^{'}\right)\right]$.

Similar 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 $x_o(t) = \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1^{'}\right)\right], \forall t \in (\tau_2, \tau_1)$. According to lemma A.2.1, offline algorithm will not turn off server during $\left(\tau_1, \tau_1^{'}\right)$. Therefore, we have $x_o(\mu) = \min\left[\bar{a}(\tau_1), \bar{a}\left(\tau_1^{'}\right)\right] = x(\mu)$.

In the two cases, we proved that $x_o(t)$ is equal to $x(t)$ con-

structed by Optimal Solution Construction Procedure. Therefore, the proposed offline algorithm can solve SCP optimally. □

A.3 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 $s_1 \neq s_2$ 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 $L(s,\cdot)$.

A.4 Proof of Theorem 5.2.2

In order to prove theorem 5.2.2, we use Lemma 5.0.2 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 $\tau^{\dagger}_{j,s} = \inf\{t > \tau_{j,e} :$ a job arrives to $s$at time $t\}$; if there are finitely many arrivals in $[0,T]$ then $\tau^{\dagger}_{j,s} = \tau_{j+1,s}$. We also consider time T as a virtual job arrival point to the server.

Lemma A.4.1. *The deterministic online ski-rental algorithm we applied in our online algorithm CSR has competitive ratio* $2-\alpha$.

Proof. As we already proved in Lemma 5.0.2, for both online and offline cases, a server faces the same set of jobs. From now on, we focus on one server, s. The server should decide to turn off itself or stay idle between $\tau_{j,e}$ and $\tau^{\dagger}_{j,s}$. In order to get competitive ratio of the deterministic online ski-rental algorithm we applied in CSR, we want to compare the power consumptions P^{on}_j and

$P_j^{off}$of the online and offline ski-rental algorithms respectively in $\left(\tau_{j,s}, \tau_{j,s}^{\dagger}\right]$. This does not include the power to turn on at $\tau_{j,s}$, but does include the power to turn on at $\tau_{j,s}^{\dagger}$ (or immediately after if $\tau_{j,s}^{\dagger}$ is an accumulation point). The power consumption of the online and offline 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

$$P_j^{off} = \begin{cases} P_b T_{j,B} + P T_{j,E}, & if\, T_{j,E} \leq \triangle; \\ P_b T_{j,B} + (\beta_{on} + \beta_{off}), & if\, T_{j,E} > \triangle. \end{cases} \tag{A.1}$$

and the online ski-rental algorithm in CSR gives

$$P_j^{on} = \begin{cases} P_b T_{j,B} + P T_{j,E}, & if\, T_{j,E} \leq \triangle; \\ P_b T_{j,B} + (\beta_{on} + \beta_{off}) + P\,(1-\alpha)\,\triangle, & if\, T_{j,E} > \triangle. \end{cases} \tag{A.2}$$

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

$$\frac{P_j^{on}}{P_j^{off}} \leq \frac{(\beta_{on}+\beta_{off}) + P(1-\alpha)\triangle}{(\beta_{on}+\beta_{off})} = 2 - \alpha.$$

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

Lemma A.4.2. *The randomized online ski-rental algorithm we applied in our online algorithm RCSR with last-empty-server-first strategy has competitive ratio* $e/(e - 1 + \alpha)$.

Proof. In the proof, we still focus on one server. We will use the same notations we used to prove Lemma A.4.1. This time

it is sufficient to compare the average power consumption P_j^{on} of the randomized online ski-rental algorithm in $\left(\tau_{j,s}, \tau_{j,s}^{\dagger}\right]$ with off-line optimal power consumption in (A.1). Under the randomized online ski-rental algorithm, when $T < \alpha\ \triangle$, we have

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

when $\alpha\ \triangle \leq T_{j,E} \leq \triangle$, we have

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

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

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

We get the above expected power consumption for $\alpha\triangle \leq T_{j,E} \leq \triangle$ as follows: If the number Z generated by the server is less than $T_{j,E} - \alpha\triangle$, 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 $Z < T_{j,E} - \alpha\triangle$. Therefore, it turns itself off and cost power $(\beta_{on} + \beta_{off})$. On the other hand, if $Z \geq T_{j,E} - \alpha\triangle$, the server will not turn itself off and consume $PT_{j,E}$ to stay idle. We can get the expected power consumption for $T_{j,E} < \alpha\triangle$ and $T_{j,E} > \triangle$ in the same way. According to the distribution of Z in RCSR, we can calculate $\mathbb{E}(P_{j,on})$ and the ratio between $\mathbb{E}(P_{j,on})$ and $P_{j,off}$:

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

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

Now we are ready to prove theorem *5.2.2.*

As we already proved in our offline algorithm that the optimal power consumption of the data center can be achieved by each server run offline ski-rental algorithm individually and independently. On the other hand, in Lemma A.4.1 and A.4.2, we proved that the power consumption of deterministic and randomized online ski-rental algorithm we applied are at most $2-\alpha$ and $\frac{e}{e-1+\alpha}$ times the power consumption of off-line ski-rental algorithm for one server. Therefore, the power consumption of our online algorithm CSR is at most $2-\alpha$ times the power consumption of offline 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 5.2.1, 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 under our job-dispatching strategy. First, we prove that the best competitive ratio of deterministic algorithm for single ski-rental problem is $2-\alpha$. Assume that deterministic online algorithm peeks into the look-ahead window and then decide to turn off or stay idle time $\theta \triangle$ after becoming empty at t_1. For $\theta < 1-\alpha$, if the server receives its next job right after $t_1 + (\theta+\alpha) \triangle$, then the online algorithm will turn off itself at $t_1 + \theta \triangle$, and consume energy $P(\theta+1) \triangle$. On the other hand, the offline optimal is $(\alpha+\theta) P \triangle$, 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 $t_1 + (\theta+\alpha) \triangle$, then the online algorithm will turn off itself at $t_1 + \theta \triangle$, and consume $P(\theta+1) \triangle$ power. On the other hand, the offline optimal is $P \triangle$. 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 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 A.4.3. *The best competitive ratio of deterministic algorithm for the repeated ski-rental problem faced by each server is* $2-\alpha$.

Proof. We will prove lemma A.4.3 by induction. Assume that deterministic algorithm $A1$ achieves the best competitive ratio for repeated ski-rental problem. Let $\theta_i \triangle$ be the length of idle time before the server peeking into the look-ahead in the ith ski-rental problem. Since the best deterministic algorithm for single ski-rental problem will peek into look-ahead window after staying idle for $(1-\alpha)\triangle$ time, thus $A1$ must have $\theta_1 = 1-\alpha$.

Suppose $\theta_i = 1-\alpha$ for $i = 1, 2, ...k$. Therefore, the cost of $A1$ is at most $2-\alpha$ times the offline optimal for the first k ski-rental problems. We will prove that $A1$ must have $\theta_{k+1} = 1-\alpha$. As a matter of fact, 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 offline optimal in worst case. It contradicts $A1$ achieves the best competitive ratio for repeated ski-rental problem. Therefore, we must have $\theta_{k+1} = 1-\alpha$. It follows that the best competitive ratio of deterministic algorithm for repeated ski-rental algorithm is $2-\alpha$. □

Therefore, the best deterministic online algorithm is CSR, which has competitive ratio $2-\alpha$.

Finally, we want to prove that RCSR has the best competitive ratio for randomized online algorithms under our last-empty-server-first job-dispatching strategy. 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 A.4.2, it is sufficient to find the minimal ratio of the power consumed by 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 $\triangle$ 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 offline cost. Thus the minimum competitive ratio satisfies

$$\inf \quad c \tag{A.3}$$

$$\text{s.t.} \quad D\sum_{i=1}^{\infty} p_i \leq cD, \quad \forall D \in [0, k], \tag{A.4}$$

$$\sum_{i=1}^{D-k} (b+i-1)\, p_i + \sum_{i=D-k+1}^{\infty} Dp_i \leq Dc, \; \forall D \in (k, b] \tag{A.5}$$

$$\sum_{i=1}^{D-k} (b+i-1)\, p_i + \sum_{i=D-k+1}^{\infty} Dp_i \leq bc, \; \forall D \in (b, \infty] \tag{A.6}$$

$$\sum_{i=1}^{\infty} p_i = 1, \qquad 0 \leq p_i \leq 1, \forall i, \tag{A.7}$$

$$\text{var} \quad c, p_i, \quad \forall i \in \{1, 2, 3, \ldots\}\,. \tag{A.8}$$

We can apply the steps in [3] to show that the optimal value c_d^* of problem (A.3)–(A.8) is equal to the optimal value $\bar{c}^*$ of following problem.

$$\begin{aligned}
\min \quad & \bar{c} && \text{(A.9)}\\
\text{s.t.} \quad & 1 \le \bar{c}, \quad \forall D \in [0,k], && \text{(A.10)}\\
& \sum_{i=1}^{D-k} (b+i-1)\,\bar{p}_i + \sum_{i=D-k+1}^{b-k} D\bar{p}_i \le D\bar{c}, \ \forall D \in (k,b), && \text{(A.11)}\\
& \sum_{i=1}^{b-k} (b+i-1)\,\bar{p}_i \le b\bar{c}, \quad \forall D \in [b,\infty]\,, && \text{(A.12)}\\
& \sum_{i=1}^{b-k} \bar{p}_i = 1, \qquad 0 \le p_i \le 1, \forall i, && \text{(A.13)}\\
\text{var} \quad & \bar{c}, \bar{p}_i, \quad \forall i \in \{1,2,\ldots,b-k\}\,. && \text{(A.14)}
\end{aligned}$$

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 (A.11)–(A.12) must hold as strict inequalities for $D \le k+j-1$, for the following reason. First consider the constraint for $D = k+j$. Since we have $j \le b-k-1$ (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

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

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

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

Since $k \le b-2$, 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 (A.10)–(A.13) 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 [3] to show that each of the inequalities in (A.11), (A.12) 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:

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

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

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

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 minimal 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 [9].

Lemma A.4.4. *The best competitive ratio of randomized algorithm for the repeated ski-rental problem faced by each server is* $\frac{e}{e-1+\alpha}$.

Proof. In the proof we will use the notation used in the proof of lemma A.4.1. Assume that the cost of online algorithm in the ith ski rental problem is C_i and the offline optimal in the ith ski rental is C_i^*. If the strategy of the oblivious adversary is to arbitrarily choose a number as the empty period in each ski rental problem, then the online algorithm has no information of the length of empty period $T_{i,E}$ of current ski rental problem even the online algorithm knows $T_{i-1,E}$, $T_{i-2,E}$,...$T_{1,E}$. Assume that online algorithm chooses $f_{Z_i}(z_i)$ as the probability distribution of Z_i given the actual values of Z_{i-1}, $T_{i-1,E}$, Z_{i-2}, $T_{i-2,E}$, ...,Z_1, $T_{1,E}$. Then there always exists a $\bar{T}_{i,E} \in (\alpha\Delta, \Delta]$ (we don't need to consider the case of $\bar{T}_{i,E} > \Delta$ because in this case the cost of online algorithm and offline algorithm are equal to that under the situation $\bar{T}_{i,E} = \Delta$) such that

$$E(C_i|Z_{i-1}, Z_{i-2}, ..., Z_1) \geq \frac{e}{e-1}\bar{T}_{i,E}.$$

To see this, suppose there is no $\bar{T}_{i,E} \in (\alpha\Delta, \Delta]$ satisfying above inequality, then for any $T_{i,E} \in (\alpha\Delta, \Delta]$, we must have

$$E\left(C_i | Z_{i-1}, Z_{i-2}, ..., Z_1\right) < \frac{e}{e-1} T_{i,E}.$$

Then in the single ski rental problem, we can also let the distribution of random variable Z follow the some distribution as Z_i. We can get a better competitive ratio than $\frac{e}{e-1}$ for single ski rental problem in this way. This is a contradiction. Therefore, such $\bar{T}_{i,E}$ must exist.

Following this, we have

$$E\left(C_i\right) = E\left(E\left(C_i | Z_{i-1}, Z_{i-2}, ..., Z_1\right)\right) \geq \frac{e}{e-1} \bar{T}_{i,E} = \frac{e}{e-1} C_i^*.$$

It is clear that the expected total cost of online algorithm is $E\left(\sum C_i\right) = \sum E\left(C_i\right)$ and the total cost of offline algorithm is $\sum C_i^*$. The competitive analysis is a worst case analysis even if the worst case dose not happen very often, and we indeed have cases in which we have $E\left(C_i\right) \geq \frac{e}{e-1} C_i^*$. Therefore, in the worst cases of the online algorithm, we must have

$$E\left(\sum C_i\right) = \sum E\left(C_i\right) \geq \frac{e}{e-1} \sum C_i^*.$$

This means the online algorithm can not do better than $\frac{e}{e-1}$ even against oblivious adversary. Therefore, RCSR has the best competitive ratio $\frac{e}{e-1+\alpha}$ for randomized algorithms against oblivious adversary. □

A.5 Proof of Corollary 5.4.1

In this section, we are going to prove corollary 5.4.1.

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 $x(t)$ 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 $x(t)$ 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 $x(t)$ 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 of 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 an 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(corresponding to case with $T_s=0$) is a lower bound of P_S^*(This is because in the optimal solution to the problem with setup time each server will be assigned a sequence of jobs. And each server tries to minimize its power cost when processing the sequence of jobs. It is obvious that the smaller T_s, the smaller the power cost of each server. Thus the total cost is smaller). 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 and $P_S^*\geq a_{\min}PT$. Since $1/\gamma$ is a lower bound of $a_{\min}$, we can directly get the upper bound of competitive ratio stated in corollary 5.4.1. □

A.6 Proof of Lemma 7.1.1

When $\theta\leq\overline{\theta}$, we have $(1-\theta)\,d\geq 1$. If the cow can see clearly things within a distance of θd, where $d\geq 1, 0\leq\theta\leq 1$, then in or-

der to find the target the cow only needs to find the point M which is $(1-\theta)\,d$ away from S. To find point M is a new traditional lost cow problem(without future information). Since our deterministic and randomized algorithms adopt the traditional algorithms for the lost cow problem to find M, the cow at most walks for a distance of $9\,(1-\theta)\,d$ and $4.59\,(1-\theta)\,d$ to reach point M. The distance from M to S is θd. Therefore the total distance is at most $(9-8\theta)\,d$ and $(4.59-3.59\theta)d$, respectively. The competitive ratios follow directly from the fact the the offline optimal distance is d.

When $\overline{\theta} < \theta < 1$, then we have $(1-\theta)\,d < 1$. Since the distance between S and M is $(1-\theta)\,d < 1$, finding point M is not a traditional lost cow problem. In this case, it is clear that the worst case for our deterministic algorithm is that the first search of the cow is in the wrong direction. Since $(1-\theta)\,d < 1$, according to our deterministic algorithm, the cow will find the target after the first failed search and the total distance traveled by the cow is $d+2$. On the other hand, in SmartCow(θ), the cow has probability 0.5 to choose the right direction and the wrong direction in the first search, respectively. And the total distance the cow has to travel is d if the direction is right and $d+2r^{\varepsilon}$ if the direction is wrong. It is easy to calculate that the expected distance traveled by the cow for SmartCow(θ) is $d+E(r^{\varepsilon})$, where ε is uniformly distributed in $[0,1)$. Since $r \approx 3.59$, we have that the value of $d+E(r^{\varepsilon}) \approx d+2$. The competitive ratios follow directly from the fact the the offline optimal distance is $d=\frac{1}{1-\overline{\theta}}$.

When $\theta = 1$, then the cow can see the target at S. Therefore, the cow can go directly to the target and this is what the offline algorithm does. In this case, the competitive ratios for both algorithms are 1.

A.7 Proof of Theorem 7.3.1

If the future information is known in the secretary problem, the probability of our strategy to hire the best applicant is:

$$\sum_{j=k+1}^{n-m} \frac{1}{n}\frac{k}{j-1} + \sum_{j=k+1}^{n-m} \frac{1}{j}\frac{k}{j-1}\frac{m}{n}. \tag{A.15}$$

In above expression, $\sum_{j=k+1}^{n-m} \frac{1}{n}\frac{k}{j-1}$ is the probability that the applicant currently being considered whether to hire or not is the best one among the total n applicants and $\sum_{j=k+1}^{n-m} \frac{1}{n}\frac{k}{j-1}$ is the probability that the best one among the total n applicants is in the m applicants whose information(considered as future information) is known by employer. The optimal k for this strategy is the one maximizing the total probability computed by (A.15). For small n, the optimal k can be easily computed. We are interested in the approximate value of the optimal k for large n.

For large n, we have $\sum_{j=k+1}^{n-m} \frac{1}{n}\frac{k}{j-1} \approx \frac{k}{n}\ln\frac{n(1-\theta)}{k}$ and $\sum_{j=k+1}^{n-m} \frac{1}{j}\frac{k}{j-1}\frac{m}{n} = \theta k\left[\frac{1}{k} - \frac{1}{n(1-r)}\right]$. Let $\frac{k}{n} = x$ and x is continuous between 0 and 1, and then the problem becomes to find the optimal x maximizing the following expression:

$$x\ln\frac{1-\theta}{x} + \theta\left[1 - \frac{x}{1-\theta}\right]. \tag{A.16}$$

It is easy to find the optimal x^* to (A.16) is $(1-\theta)\,e^{-\frac{1}{1-\theta}}$ and the corresponding probability is $\theta+(1-\theta)\,e^{-\frac{1}{1-\theta}}$. It indicates that the probability of hiring the best applicant is enhanced. Moreover,

If $\theta = 0$, we have $x^* = \frac{1}{e}$ and the probability is $\frac{1}{e}$ which match the results from the classic secretary problem.

□ **End of chapter.**

Bibliography

[1] Spain Energy Consumption, http://www.nationmaster.com/country/sp-spain/ene-energy.

[2] SPEC power data, http://www.spec.org.

[3] Online algorithms: Ski rental, Claire Mathieu, http://www.cs.brown.edu/∼claire/Talks/skirental.pdf.

[4] 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.

[5] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106:234–234, 1993.

[6] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. 2004.

[7] L. Barroso. The price of performance. *ACM Queue*, 3(7):48–53, 2005.

[8] L. Barroso and U. Holzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[9] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

[10] 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*.

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

[12] 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.

[13] 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*, volume 33, pages 303–314, 2005.

[14] 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.

[15] 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.

[16] T. Ferguson. Who solved the secretary problem? *Statistical science*, pages 282–289, 1989.

[17] H. Fujiwara and K. Iwama. Average-case competitive analyses for ski-rental problems. *Algorithms and Computation*, pages 157–189, 2002.

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

[19] J. Gilbert and F. Mosteller. Recognizing the maximum of a sequence. *Selected Papers of Frederick Mosteller*, pages 355–398, 2006.

[20] 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*, pages 186–195. ACM, 2005.

[21] P. Jaillet and M. Wagner. Online routing problems: Value of advanced information as improved competitive ratios. *Transportation Science*, pages 200–210, 2006.

[22] T. Jayram, T. Kimbrel, R. Krauthgamer, B. Schieber, and M. Sviridenko. Online server allocation in a server farm via benefit task systems. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 540–549, 2001.

[23] M. Kao, J. Reif, and S. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 441–447, 1993.

[24] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, 1994.

[25] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

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

[27] 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*, 41(1):102–108, 2011.

[28] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[29] L. X. L. Rao, X. Liu and W. Liu. Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricitymarket environment. *Proc. IEEE INFOCOM*, 2010.

[30] M. Lin, Z. Liu, A. Wierman, and L. L. H. Andrew. Online algorithms for geographical load balancing. In *Proc. Int. Green Computing Conf.*, 2012.

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

[32] Z. Liu. *Greening geographical load balancing.* PhD thesis, California Institute of Technology, 2011.

[33] A. Mądry and D. Panigrahi. The semi-stochastic ski-rental problem.

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

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

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

[37] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage (TOS)*, 4(3):10, 2008.

[38] 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*, pages 5–5. IEEE, 2007.

[39] V. Petrucci, O. Loques, and D. Mossé. Dynamic optimization of power and performance for virtualized server clusters. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 263–264, 2010.

[40] 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.

[41] K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. *Algorithm Theory-SWAT 2004*, pages 14–25, 2004.

[42] 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*, pages 4–4, 2011.

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

[44] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 48–59, 2008.

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

[46] 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.

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

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

[49] P. Wendell, J. Jiang, M. Freedman, and J. Rexford. Donar: decentralized server selection for cloud services. In *Proc. ACM SIGCOMM*, volume 40, pages 231–242, 2010.

[50] A. Wierman, L. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems. In *INFOCOM 2009, IEEE*, pages 2007–2015. IEEE, 2009.

[51] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 374–382. IEEE, 1995.