# Algorithmic Challenges in Green Data Centers

Thesis by

Minghong Lin

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy


California Institute of Technology

Pasadena, California

2013

(Defended June 6, 2013)

*This thesis is dedicated to*
*my wife Xuefang,*
*whose love made this thesis possible,*
*and my parents,*
*who have supported me all the way.*

# Acknowledgements

First, I would like to express my deepest gratitude to my advisor, Adam Wierman, for his thoughtful guidance, insightful vision and continuing support. His patience and encouragement helped me overcome many crisis situations. I am thankful for the opportunity to learn from him.

Next, I am grateful to my collaborators, Lachlan Andrew, Bert Swart, Eno Thereska, Steven Low, Zhenhua Liu, Lijun Chen, Jian Tan and Li Zhang. It is a great pleasure to work with them. They have always provided insightful discussions and constructive suggestions. I am also thankful to my former research advisors, John C.S. Lui and Dah-Ming Chiu, for their guidance during my M.Phil study in Hong Kong. They were the reason why I decided to go to pursue a Ph.D. I would also like to take this opportunity to express my gratitude to Mani Chandy. His comments and questions were very beneficial in my completion of the thesis.

I greatly enjoyed the opportunity to study in Computer Science at Caltech, which provides amazing supportive environment for students. It is wonderful to have so many intelligent professors and outstanding students around to ask for advice and opinions. I would also like to thank the helpful administrative staff in our department, especially Sydney Garstang.

Finally, I wish to thank my family for providing a loving environment for me. My parents and my brother receive my deep gratitude for their dedication and the many years of support. I would like to thank my wife Xuefang for her understanding and love during the past few years. Her support and encouragement was what made this thesis possible.

# Abstract

With data centers being the supporting infrastructure for a wide range of IT services, their efficiency has become a big concern to operators, as well as to society, for both economic and environmental reasons. The goal of this thesis is to design energy-efficient algorithms that reduce energy cost while minimizing compromise to service. We focus on the algorithmic challenges at different levels of energy optimization across the data center stack. The algorithmic challenge at the device level is to improve the energy efficiency of a single computational device via techniques such as job scheduling and speed scaling. We analyze the common speed scaling algorithms in both the worst-case model and stochastic model to answer some fundamental issues in the design of speed scaling algorithms. The algorithmic challenge at the local data center level is to dynamically allocate resources (e.g., servers) and to dispatch the workload in a data center. We develop an online algorithm to make a data center more power-proportional by dynamically adapting the number of active servers. The algorithmic challenge at the global data center level is to dispatch the workload across multiple data centers, considering the geographical diversity of electricity price, availability of renewable energy, and network propagation delay. We propose algorithms to jointly optimize routing and provisioning in an online manner. Motivated by the above online decision problems, we move on to study a general class of online problem named "smoothed online convex optimization", which seeks to minimize the sum of a sequence of convex functions when "smooth" solutions are preferred. This model allows us to bridge different research communities and help us get a more fundamental understanding of general online decision problems.

# Contents

# Chapter 1

# Introduction

Data centers provide the supporting infrastructure for a wide range of IT services and consume a significant amount of electricity. According to the US EPA Report to the Congress on Server and Data Center Energy Efficiency in 2007, US data centers consume 61 billion kWh in 2006 (1.5% of total U.S. electricity). Moreover, it is growing exponentially at an annual rate of 15%. A recent report [76] revealed that although the growth rate slowed down a little recently, electricity used by data centers worldwide increased by about 56% from 2005 to 2010. Further, from an operator's stand point, the energy cost has grown to exceed the server costs in data centers. Thus, it is not surprising that optimizing energy cost in data center is receiving increasing attention. However, saving energy and improving performance are usually in conflict with each other, and thus the joint optimization is a challenge.

## 1.1 Energy efficiency of data centers

Traditionally computer systems usually focused on performance and throughput. Energy efficiency as a new focus has been studied in the mobile and embedded areas due to the limited battery capacity. As the IT becomes a significant consumer of energy resources and a substantial source of greenhouse gas pollution, the focus on energy-efficient computing has expanded to general-purpose computing over the past decade, including servers and data centers. Many energy-saving techniques developed for mobile devices directly benefit the design of energy-efficient servers, such as clock gating and DVFS technique. However there are also many differences. For example, the activity pattern for mobile devices is very different than that of the servers [16]. Mobile devices need high performance for short periods and then remain idle for a long periods. Therefore, the design of mobile devices can focus on the energy efficiency in peak performance mode and idle mode. On the other hand, servers operate at a mild load level for most of the time, i.e., they are rarely idle or operating at maximum load. As a result, servers do not benefit much from inactive low-energy states like sleep or standby that work very well for mobile devices. Instead, speed-scaling technique

may be more useful for servers.

For a data center, besides the energy consumed by the servers performing the computation, a large fraction of energy is consumed by the cooling and provisioning infrastructure. To capture this consumption, power usage effectiveness (PUE) measures the ratio of total building power to IT power, i.e., the power consumed by the actual computing equipments such as servers, network equipments and so on. It is reported that PUE was greater than 2 for typical data centers [52].

Fortunately, PUE can be substantially improved by careful design for energy efficiency. The most energy-efficient data centers today have PUE $\leq$ 1.2 [17]. This is achieved via a few steps (a) maintaining data centers at a higher temperature. It has been shown that increasing the cold aisle temperatures to 25-27$^{\circ}C$ can save a large amount of cooling energy without causing higher equipment failures. (b) using more efficient air-flow control to reduce the energy needed for cooling. This is one of the primary reasons why container-based data centers are more efficient, because the hot air is isolated from the cold air and the path to the cooling coil is short. (c) adopting more efficient gear to reduce the UPS and power distribution losses. For example, using per-server UPSs instead of a facility-wide UPS will increase the efficiency of the overall power infrastructure by eliminating the AC-DC-AC overhead.

Beyond these engineering improvements, there is also significant energy reduction to be achieved via improved IT design in data centers [18, 3]. From the hardware perspective, according to [95], the main part of the power in a server is consumed by CPU, followed by the memory and the power loss. However, CPU no longer dominates the power consumption of the server. This is because the modern CPUs are adopting much more energy-efficient techniques than other system components. As a result, they can consume less than 30% of their peak power in low-activity mode, i.e., the dynamic power range is more than 70% of the peak power [16]. In contrast, the dynamic power ranges of all other components are much narrower: less than 50% for DRAM, 25% for disk drives, 15% for network switches, and negligible for other components. The energy efficiency of IT components has been widely studied from an algorithmic perspective as well. The goal here is usually to design energy-efficient algorithms that reduce energy consumption while minimizing compromise to service.

## 1.2 Algorithmic challenges in energy efficiency

Generally the literature on energy-efficient algorithms can be classified into three categories based on the "layer" of the data center that is the focus: server level, local data center level and global data center level. The optimization at the server level is to improve the energy efficiency of a single server via techniques such as scheduling and speed scaling. The optimization at the local data center level is to decide how many resources (e.g., servers) to use and how to dispatch the workload among

servers. The optimization at the global data center level is to dispatch the workload across multiple data centers, considering electricity price diversity and propagation delay diversity. We focus on the algorithmic challenges at all three levels in this thesis.

At the server level, many energy-saving techniques developed for mobile devices directly benefit the design of energy-efficient servers such as power-down mechanism and speed-scaling technique. However, because of the activity pattern, speed-scaling technique is more useful than power-down mechanism for servers. The basic idea behind speed scaling is that running at a low speed consumes less energy. But running at a low speed will make the user delay increase. Thus the speed scaling algorithms need to make a tradeoff between performance and energy usage. Depending on the objective, we may want to minimize the energy usage while meeting job deadlines, optimize user experience given energy budget, or minimize a linear combination of user delay and energy usage.

At the local data center level, a guiding focus for research into 'green' data centers is the goal of designing data centers that are 'power-proportional', i.e., use power only in proportion to the load. However, current data centers are far from this goal – even today's energy-efficient data centers consume almost half of their peak power when nearly idle [16]. A promising approach for making data centers more power-proportional is using software to dynamically adapt the number of active servers to match the current workload, i.e., to dynamically 'right-size' the data center. The algorithmic question is, how to determine how many servers to be active and how to control servers and requests.

At the global data center level, as the demand on Internet services has increased in recent years, enterprises have move to using several distributed data centers to provide better QoS for users. To improve user experience, they tend to disperse data centers geographically so that user requests from different regions can be routed to data centers nearby, thus reducing the propagation delay. Recently, since the energy cost is becoming a big fraction for the total cost of the data centers, it has been proposed that the energy costs, both monetary and environmental, can be reduced by exploiting temporal variations and shifting processing to data centers located in regions where energy currently has low cost. Lightly loaded data centers can then turn off surplus servers.

## 1.3 Overview of this thesis

This thesis is divided into four components. In Chapter 2 we focus on the speed scaling problem at the server level. In Chapter 3 we study the capacity management problem at the local data center level. In Chapter 4 we investigate the geographical load balancing problem at the global data center level. Finally, in Chapter 5 we move beyond the data center area and study a general optimization framework for online decision problems.[1]

[1]Note that the notations in different chapters are independent.

**Chapter 2: Server speed scaling**

Algorithmic work at the server level focuses on designing algorithms to reduce energy consumption while minimizing compromise to performance. Most of the algorithms studied are online algorithms since the device has to decide which action to take at the current time without knowing the future.

The algorithmic questions that have been studied most widely at the server level are power-down mechanisms and speed scaling. Our focus is on speed scaling algorithms, but we begin by briefly surveying power-down mechanisms.

Power-down mechanisms are widely used in mobile devices, e.g., laptop goes to sleep mode if it has been idle longer than a certain threshold. The design question is how to determine such idle thresholds. Generally, a device has multiple states, each state has its own power consumption rate, and it consumes a certain amount of energy to transit from one state to others. The device must be at active state to serve tasks, and it may go to some sleep states during idle periods to save energy. The goal is to minimize the total energy. It has been shown that the energy consumed by the best possible deterministic online algorithm is at most twice that of the optimal solution, and randomized algorithms can do even better [65]. Many generalizations of this problem have been studied, including stochastic settings [10].

Speed scaling is another way to save energy for variable speed devices, since running at a low speed consumes less energy. Fundamentally, a speed scaling algorithm must make two decisions at each time:(i) a *scheduling policy* must decide which job(s) to service, and (ii) a *speed scaler* must decide how fast to run the server. The analytic study of the speed scaling problem began with Yao et al. [124] in 1995. Since [124], three main performance objectives balancing energy and delay have been considered: (i) minimize the total energy used in order to meet job deadlines [14, 101], (ii) minimize the average response time given an energy budget [32, 125], and (iii) minimize a linear combination of expected response time and energy usage per job [4, 13].

Despite the considerable algorithmic literature, there are many fundamental issues in the design of speed scaling algorithms that are not yet understood. Can a speed scaling algorithm be optimal? What structure do (near-)optimal algorithms have? How does speed scaling interact with scheduling? How important is the sophistication of the speed scaler? What are the drawbacks of speed scaling?

Our results show that *"energy-proportional" speed scaling provides near-optimal performance*, i.e., running at the speed such that the power is proportional to the number of jobs in the system. Additionally, we show that *speed scaling can be decoupled from the scheduler*. That is, energy-proportional speed scaling performs well for the common scheduling policies. Further, our results show that scheduling is not as important once energy is considered, i.e., policies that differ greatly when optimizing for delay have nearly the same performance when energy is considered. Our results highlight that the optimal gated-static speed scaling algorithm performs nearly as well as the optimal dynamic speed scaling algorithm. Thus, *sophistication does not provide significant performance*

*improvements in speed scaling designs.* Finally, our results uncover one unintended drawback of dynamic speed scaling: *speed scaling can magnify unfairness.*

The work presented in this chapter is based on the publications [7, 85].

**Chapter 3: Dynamic capacity provisioning in data centers**

Algorithmic questions at the local data center level focus on allocating compute resources for incoming workloads and dispatching workloads in the data center. The goal of design is to achieve "energy proportionality" [16], i.e., use power only in proportion to the load. A promising approach for making data centers more power-proportional is to dynamically 'right-size' the data center. Specifically, dynamic right-sizing refers to adapting the way requests are dispatched to servers in the data center so that, during periods of low load, servers that are not needed do not have jobs routed to them and thus are allowed to enter power-saving modes (e.g., go to sleep or shut down).

Technologies that implement dynamic right-sizing are still far from standard in data centers due to a number of challenges. First, servers must be able to seamlessly transition into and out of power-saving modes while not losing their state. There has been a growing amount of research into enabling this in recent years, dealing with virtual machine state [39], network state [37] and storage state [108, 5]. Second, such techniques must prove to be reliable, since administrators may worry about wear-and-tear consequences of such technologies. Third, it is unclear how to determine how many servers to toggle into power-saving mode and how to control servers and requests.

We provide a new algorithm to address this third challenge. We develop a simple but general model that captures the major issues that affect the design of a right-sizing algorithm, including: the cost (lost revenue) associated with the increased delay from using fewer servers, the energy cost of maintaining an active server with a particular load, and the cost incurred from toggling a server into and out of a power-saving mode (including the delay, energy, and wear-and-tear costs). First, we analytically characterize the optimal solution. We prove that it exhibits a simple, 'lazy' structure when viewed in reverse time. Second, we introduce and analyze a novel, practical online algorithm motivated by this structure, and prove that this algorithm guarantees cost no larger than 3 times the optimal cost, under very general settings  arbitrary workloads, and general delay cost and general energy cost models provided that they result in a convex operating cost. Further, in realistic settings its cost is nearly optimal. Additionally, the algorithm is simple to implement in practice and does not require significant computational overhead. Moreover, we contrast it with the more traditional approach of receding horizon control and show that our algorithm provides much more stable cost saving with general settings.

Furthermore, we validate our algorithm using two load traces (from Hotmail and a Microsoft Research data center) to evaluate the cost savings achieved via dynamic right-sizing in practice. We show that significant savings are possible under a wide range of settings.

The work presented in this chapter is based on the publications [83].

**Chapter 4: Cost-effective geographical load balancing**

The algorithmic questions at the global data center level focus on exploring the diversity of power prices and the diversity of propagation delays given geographically distributed data centers. Further, electricity prices and workloads are time-varying, which makes the joint optimization on energy and performance even more challenging. There is a growing literature related to the energy optimization of geographic dispersed data centers, but is still a fairly open problem. So far, [104] investigates the problem of total electricity cost for data centers in multi-electricity-market environment and propose a linear programming formulation to approximate it. They consider the queueing delay constraint inside the data center (assumed to be an $M/M/1$ queue) but not the end-to-end delay of users, thus the diversity of propagation delay has not been explored. Another approach, DONAR [118] runs a simple, efficient algorithm to coordinate their replica-selection decisions for clients with the capacity at each data center fixed. The distributed algorithm solves an optimization problem that jointly considers both client performance and server load. However, DONAR does not optimize the capacity provision at each data center and thus does not explore the diversity of power price. Moreover, neither approach considers the time variation of power price and workloads in the optimization problem.

We developed a framework to jointly optimize the total energy cost and the end-to-end delay of users by considering the price diversity and delay diversity. Our goal is to find a global dispatching scheme to route the workload from different regions to certain data centers dynamically, while considering the capacity optimization at each data center. Similar to the energy optimization at the local data center level, we would like to optimize both the operating cost for providing the service and the switching cost for changing the provisioning in an online manner. A commonly suggested algorithm for this setting is "receding horizon control" (RHC), which computes the provisioning for the current time by optimizing over a window of predicted future loads. We show that RHC performs well in a homogeneous setting, in which all servers can serve all jobs equally well; however, we also prove that differences in propagation delays, servers, and electricity prices can cause RHC to perform badly. So, we introduce variants of RHC that are guaranteed to perform as well in the face of such heterogeneity.

We then uses these algorithms to study the environmental potential of geographical load balancing. We illustrate that the geographical diversity of Internet-scale services can significantly improve the efficiency of the usage of renewable energy and this potential can be realized by our online algorithms. The numerical experiments show that using our algorithms for "follow the renewables" routing can provide significant environmental benefits. These algorithms are then used to study the feasibility of powering a continent-wide set of data centers mostly by renewable sources, and to

understand what portfolio of renewable energy is most effective. The numerical results reveal that the optimal renewable portfolio may include more wind power than solar power, though solar power seems to have better correlation with the workload shape.

The work presented in this chapter is based on the publications [82, 88, 87].

**Chapter 5: Smoothed online convex optimization**

The optimization problems in Chapter 3 and Chapter 4 share a similar property: We would like to adapt our decision (e.g., capacity, routing) based on the time-varying environment (e.g., workload, electricity price, renewable availability), but we do not want to adapt it too frequently because changing the decisions incurs overhead (e.g., service migration, data movement, wear-and-tear consequence). Actually this property also exists in many other problems even outside of the data center environment. For example, video streaming [67], where the encoding quality of a video needs to change dynamically in response to network congestion, but where large changes in encoding quality are visually annoying to users; optical networking [126], in which there is a cost to reestablish a light path; and content placement problems for CDN, in which there is a cost to move data. In addition to applications within computer systems, there are a number of problems in industrial optimization having similar property. One is the dynamic dispatching of electricity generators, where a particular portfolio of generation must be allocated to cover demand, but in addition to the time-varying operating costs of the generators there are significant "setup" and "ramping" costs associated with changing the generator output [69].

In this chapter we consider the general "smoothed online convex optimization" (SOCO) problems, a variant of the class of online convex optimization (OCO) problems that is strongly related to metrical task systems. Actually, many applications typically modeled using online convex optimization have, in reality, some cost associated with a change of action; and so may be better modeled using SOCO rather than OCO. For example, OCO encodes the so-called "$k$-experts" problem, which has many applications where switching costs can be important, e.g., in stock portfolio management there is a cost associated with adjusting the stock portfolio owned. In fact, "switching costs" have long been considered important in related learning problems, such as the $k$-armed bandit problem which has a considerable literature studying algorithms that can learn effectively despite switching costs [9, 54]. Further, SOCO has applications even in contexts where there are no costs associated with switching actions. For example, if there is concept drift in a penalized estimation problem, then it is natural to make use of a regularizer (switching cost) term in order to control the speed of the drift of the estimator.

Prior literature on these problems has focused on two performance metrics: regret and the competitive ratio. There exist known algorithms with sublinear regret and known algorithms with constant competitive ratios; however, no known algorithm achieves both simultaneously. We show

that this is due to a fundamental incompatibility between these two metrics – no algorithm (deterministic or randomized) can achieve sublinear regret and a constant competitive ratio, even in the case when the objective functions are linear. However, we also exhibit an algorithm that, for the important special case of one-dimensional decision spaces, provides sublinear regret while maintaining a competitive ratio that grows arbitrarily slowly.

The work presented in this chapter is based on the publications [84, 6].

# Chapter 2

# Server Speed Scaling

Computer systems must make a fundamental tradeoff between performance and energy usage. The days of "faster is better" are gone — energy usage can no longer be ignored in designs, including chips and servers. The importance of energy has led the designs to move toward speed scaling. Speed scaling designs adapt the "speed" of the system so as to balance energy and performance measures. Speed scaling designs can be highly sophisticated — adapting the speed at all times to the current state (*dynamic speed scaling*) — or very simple — running at a static speed that is chosen *a priori* to balance energy and performance, except when idle (*gated-static speed scaling*).

The growing adoption of speed scaling designs for systems from chips to disks has spurred analytic research into the topic. The analytic study of the speed scaling problem began with Yao et al. [124] in 1995. Since [124], three main performance objectives balancing energy and delay have been considered: (i) minimize the total energy used in order to meet job deadlines, e.g., [14, 101] (ii) minimize the average response time given an energy/power budget, e.g., [32, 125], and (iii) minimize a linear combination of expected response time and energy usage per job [4, 13]. We focus on the third objective. This objective captures how much reduction in response time is necessary to justify using an extra 1 joule of energy, and naturally applies to settings where there is a known monetary cost to extra delay (e.g., many web applications). This is related to (ii) by duality.

Fundamentally, a speed scaling algorithm must make two decisions at each time: (i) a *scheduling policy* must decide which job(s) to service, and (ii) a *speed scaler* must decide how fast to run the server. It has been noted by prior work, e.g., [101], that an optimal speed scaling algorithm will use shortest remaining processing time (SRPT) scheduling. However, in real systems, it is often impossible to implement SRPT, since it requires exact knowledge of remaining sizes. Instead, typical system designs often use scheduling that is closer to processor sharing (PS), e.g., web servers, operating systems, and routers. We focus on the design of speed scalers for both SRPT and PS.

The study of speed scaling algorithms for these two policies is not new. There has been significant prior work, which we discuss in Sections 2.2.1 and 2.2.2, studying speed scaling for SRPT [4, 12, 13, 15, 81] and for PS [30, 42, 48, 117, 120]. Interestingly, the prior work for SRPT is entirely done

using a worst-case framework while the prior work for PS is done in a stochastic environment, the M/GI/1 queue.

Despite the considerable literature studying speed scaling, there are many fundamental issues in the design of speed scaling algorithms that are not yet understood. This work provides new insights into four of these issues:

I *Can a speed scaling algorithm be optimal? What structure do (near-)optimal algorithms have?*

II *How does speed scaling interact with scheduling?*

III *How important is the sophistication of the speed scaler?*

IV *What are the drawbacks of speed scaling?*

To address these questions we study both PS and SRPT scheduling under both dynamic and gated-static speed scaling algorithms. Our work provides (i) new results for dynamic speed scaling with SRPT scheduling in the worst-case model, (ii) the first results for dynamic speed scaling with PS scheduling in the worst-case model, (iii) the first results for dynamic speed scaling with SRPT scheduling in the stochastic model, (iv) the first results for gated-static speed scaling with SRPT in the stochastic model, and (v) the first results identifying unfairness in speed scaling designs. Table 2 summarizes these.

These results lead to important new insights into Issues I-IV above. We describe these insights informally here and provide pointers to the results in the body of the chapter.

With respect to **Issue I**, our results show that *"energy-proportional" speed scaling provides near-optimal performance.* Specifically, we consider the algorithm which uses SRPT scheduling and chooses $s_n$, the speed to run at given $n$ jobs, to satisfy $P(s_n) = n\beta$ (where $P(s)$ is the power needed to run at speed $s$ and $1/\beta$ is the cost of energy). We prove that this algorithm is $(2+\varepsilon)$-competitive under general $P$ (Corollary 2.1). This provides a tight analysis of an algorithm with a considerable literature, e.g., [4, 12, 13, 15, 81] (see Section 2.2.1 for a discussion). It also gives analytic justification for a common heuristic applied by system designers, e.g., [16]. Further, we show that no "natural" speed scaling algorithm (Definition 2.1) can be better than 2-competitive (Theorem 2.2), which implies that no online energy-proportional speed scaler can match the offline optimal.

With respect to **Issue II**, our results uncover two new insights. First, we prove that, at least with respect to PS and SRPT, *speed scaling can be decoupled from the scheduler.* That is, energy-proportional speed scaling performs well for both SRPT and PS (and another policy LAPS studied in [34]). Specifically, we show that PS scheduling with speeds such that $P(s_n) = n$, which are optimally competitive under SRPT, is again $O(1)$-competitive[1] (Theorem 2.3). Further, we show

[1] $O(\cdot)$ and $o(\cdot)$ are defined in [20]; $f = \omega(g) \Leftrightarrow g = o(f)$; $f = \Omega(g) \Leftrightarrow g = O(f)$; $f = \Theta(g) \Leftrightarrow [f = O(g)$ and $g = O(f)]$.

that using the speeds optimal for an M/GI/1 PS queue to control instead an M/GI/1 SRPT queue leads to nearly optimal performance (Section 2.2.2). Second, our results show that scheduling is not as important once energy is considered. Specifically, PS is $O(1)$-competitive for the linear combination of energy and response time; however, when just mean response time is considered PS is $\Omega(\nu^{1/3})$-competitive for instances with $\nu$ jobs [96]. Similarly, we see in the stochastic environment that the performance under SRPT and PS is almost indistinguishable (e.g., Figure 2.1). Together, the insights into Issue II provide a significant simplification of the design of speed scaling systems: they suggest that practitioners can separate two seemingly coupled design decisions and deal with each individually.

With respect to **Issue III**, our results add support to an insight suggested by prior work. Prior work [120] has shown that the optimal gated-static speed scaling algorithm performs nearly as well as the optimal dynamic speed scaling algorithm in the M/GI/1 PS setting. Our results show that the same holds for SRPT (Section 2.3). Thus, *sophistication does not provide significant performance improvements in speed scaling designs.* However, sophistication provides improved robustness (Section 2.4). To support this analytically, we provide worst-case guarantees on the (near) optimal stochastic speed scalers for PS and SRPT (Corollary 2.3). Note that it is rare to be able to provide such guarantees for stochastic control policies. The insights related to Issue III have an interesting practical implication: instead of designing "optimal" speeds it may be better to design "optimally robust" speeds, since the main function of dynamic speed scaling is to provide robustness. This represents a significant shift in approach for stochastic speed scaling design.

With respect to **Issue IV**, our results uncover one unintended drawback of dynamic speed scaling: *speed scaling can magnify unfairness.* Unfairness in speed scaling designs has not been identified previously, but in retrospect the intuition behind it is clear: If a job's size is correlated with the occupancy of the system while it is in service, then dynamic speed scaling will lead to differential service rates across job sizes, and thus unfairness. We prove that speed scaling magnifies unfairness under SRPT (Theorem 2.5) and all non-preemptive policies, e.g., FCFS (Proposition 2.6). In contrast, PS is fair even with dynamic speed scaling (Proposition 2.5). Combining these results with our insights related to Issue II, we see that designers can decouple the scheduler and the speed scaler when considering performance, but should be wary about the interaction when considering fairness.

Our results highlight the balancing act a speed scaling algorithm must perform in order to achieve the three desirable properties: near-optimal performance, robustness, and fairness. It is possible to be near-optimal and robust using SRPT scheduling and dynamic speed scaling, but this creates unfairness. SRPT can be fair and still near-optimal if gated-static speed scaling is used, but this is not robust. On the other hand, dynamic speed scaling with PS can be fair and robust but, in the worst case, pays a significant performance penalty (though in stochastic settings is nearly optimal).

| Name | Scheduler | Speed scaler: $s_n$ | $P(s)$ | Optimal? | Robust? | Fair? |
|---|---|---|---|---|---|---|
| SRPT-INV | SRPT | Dynamic: $P^{-1}(n\beta)$ | General | 2-competitive (Theorem 2.1). | yes | no |
| SRPT-DP | SRPT | Dynamic: Prop. 2.1 | $s^\alpha$ | $O(1)$-competitive for $\alpha \leq 2$ (Corollary 2.3). | yes | no |
| SRPT-LIN | SRPT | Dynamic: $n\sqrt{\beta}$ | $s^2$ | No guarantee, simulation results in Figure 2.7. | weakly | no |
| SRPT-GATED | SRPT | Gated: (2.20) | Regular | $O(1)$-competitive in M/GI/1 under heavy traffic with $P(s) = s^2$ (Corollary 2.2). Optimal gated in M/GI/1 under heavy traffic (Theorem 2.4). | no | yes |
| PS-INV | PS | Dynamic: $P^{-1}(n\beta)$ | $s^\alpha$ | $O(1)$-competitive (Theorem 2.3). | yes | yes |
| PS-DP | PS | Dynamic: Prop. 2.1 | $s^\alpha$ | $O(1)$-competitive for $\alpha \leq 2$ (Corollary 2.3). Optimal in M/GI/1 PS [120]. | yes | yes |
| PS-LIN | PS | Dynamic: $n\sqrt{\beta}$ | $s^2$ | $O(1)$-competitive in M/GI/1 with $P(s) = s^2$ [120]. | weakly | yes |
| PS-GATED | PS | Gated: (2.17) | Regular | $O(1)$-competitive in M/GI/1 with $P(s) = s^2$ (Corollary 2.2). Optimal gated in M/GI/1 [120]. | no | yes |

Table 2.1: Summary of the speed scaling schemes in this chapter.

Thus, the policies considered in this chapter *can achieve any two of near-optimal, fair, and robust — but not all three.*

Finally, it is important to note that the analytic approach in this chapter is distinctive. It is unusual to treat both stochastic and worst-case models together; and further, many results depend on a combination of worst-case and stochastic techniques, which leads to insights that could not have been attained by focusing on one model alone.

## 2.1 Model and notation

We consider the joint problem of speed scaling and scheduling in a single-server queue to minimize a linear combination of expected response time (also called sojourn time or flow time), denoted by $T$, and energy usage per job, $\mathcal{E}$:

$$z = \mathbb{E}[T] + \mathbb{E}[\mathcal{E}]/\beta. \tag{2.1}$$

By Little's law, this may be more conveniently expressed as

$$\lambda z = \mathbb{E}[N] + \mathbb{E}[P]/\beta$$

where $N$ is the number of jobs in the system and $P = \lambda\mathcal{E}$ is the power expended.

Before defining the speed scaling algorithms, we need some notation. Let $n(t)$ be the number of jobs in the system at time $t$ and $s(t)$ be the speed that the system is running at time $t$. Further, define $P(s)$ as the power needed to run at speed $s$. Then, the energy used by time $t$ is $\mathcal{E}(t) = \int_0^t P(s(\tau))d\tau$.

Measurements have shown that $P(s)$ can take on a variety of forms depending on the system being studied; however, in many applications a low-order polynomial form provides a good approximation, i.e., $P(s) = ks^\alpha$ with $\alpha \in (1, 3)$. For example, for dynamic power in CMOS chips $\alpha \approx 1.8$ is a good approximation [120]. However, this polynomial form is not always appropriate. Some of our results assume a polynomial form to make the analysis tractable, and particularly $\alpha = 2$ provides a simple example which we use for many of our numerical experiments. Other results hold for general, even non-convex and discontinuous, power functions. Additionally, we occasionally limit our results to *regular* power functions, which are differentiable on $[0, \infty)$, strictly convex, non-negative, and 0 at speed 0.

Now, we can define a speed scaling algorithm: A speed scaling algorithm $\mathcal{A} = (\pi, \Sigma)$, is a pair of a scheduling discipline $\pi$ that defines the order in which jobs are processed, and a speed scaling rule $\Sigma$ that defines the speed as a function of system state, in terms of the power function, $P$. In this chapter we consider speed scaling rules where the speed is a function of the number of jobs in the system, i.e., $s_n$ is the speed when the occupancy is $n$.[2]

The scheduling algorithms $\pi$ we consider are online, and so are not aware of a job $j$ until it arrives at time $r(j)$, at which point $\pi$ learns the size of the job, $x_j$. We consider a preempt-resume model, that is, the scheduler may preempt a job and later restart it from the point it was interrupted without any overhead. The policies that we focus on are: shortest remaining processing time (SRPT), which preemptively serves the job with the least remaining work, and processor sharing (PS), which shares the service rate evenly among the jobs in the system at all times.

The speed scaling rules, $s_n$, we consider can be *gated-static*, which runs at a constant speed while the system is non-idle and sleeps while the system is idle, i.e., $s_n = s_{gs}1_{n\neq 0}$; or more generally *dynamic* $s_n = g(n)$ for some function $g : \mathbb{N} \cup \{0\} \to [0, \infty)$. Note that the speed is simply the rate at which work is completed, i.e., a job of size $x$ served at speed $s$ will complete in time $x/s$. To avoid confusion, we occasionally write $s_n^\pi$ as the speed under policy $\pi$ when the occupancy is $n$. The queue is single-server in the sense that the full speed $s_n$ can be devoted to a single job.

We analyze the performance of speed scaling algorithms in two different models — one worst-case and one stochastic.

[2]This suits objective (2.1); e.g., it is optimal for an isolated batch arrival, and the optimal $s$ is constant between arrival/departures. For other objectives, it is better to base the speed on the unfinished work instead [15].

**Notation for the worst-case model**

In the worst-case model we consider finite, arbitrary (maybe adversarial) deterministic instances of arriving jobs. A problem instance consists of $\nu$ jobs, with the $j$th job having arrival time (release time) $r(j)$ and size (work) $x_j$. Our objective is again a linear combination of response time and energy usage. Let $\mathcal{E}(I)$ be the total energy used to complete instance $I$, and $T_j$ be the response time of job $j$, the completion time minus the release time. The analog of (2.1) is to replace the ensemble average by the sample average, giving the cost of an instance $I$ under a given algorithm $\mathcal{A}$ as

$$z^{\mathcal{A}}(I) = \frac{1}{\nu}\left(\sum\nolimits_j T_j + \frac{1}{\beta}\mathcal{E}(I)\right).$$

In this model, we compare the cost of speed scaling algorithms to the cost of the optimal offline algorithm, OPT. In particular, we study the competitive ratio, defined as

$$CR = \sup_I z^{\mathcal{A}}(I)/z^O(I),$$

where $z^O(I)$ is the optimal cost achievable on $I$. A scheme is "$c$-competitive" if its competitive ratio is at most $c$.

**Notation for the stochastic model**

In the stochastic model, we consider an M/GI/1 (or sometimes GI/GI/1) queue with arrival rate $\lambda$. Let $X$ denote a random job size with c.d.f. $F(x)$, c.c.d.f. $\bar{F}(x)$, and continuous p.d.f. $f(x)$. Let $\rho = \lambda\mathbb{E}[X] \in [0, \infty)$ denote the load of arriving jobs. Note that $\rho$ is not the utilization of the system and that many dynamic speed scaling algorithms are stable for all $\rho$. When the power function is $P(s) = s^\alpha$, it is natural to use a scaled load, $\gamma := \rho/\beta^{1/\alpha}$, which jointly characterizes the impact of $\rho$ and $\beta$ (see [120]).

Denote the response time of a job of size $x$ by $T(x)$. We consider the performance metric (2.1) where the expectations are averages per job. In this model the goal is to optimize this cost for a specific workload, $\rho$. Define the competitive ratio in the M/GI/1 model as

$$CR = \sup_{F,\lambda} z^{\mathcal{A}}/z^O$$

where $z^O$ is the average cost of the optimal offline algorithm.

## 2.2 Dynamic speed scaling

We start by studying the most sophisticated speed scaling algorithms, those that dynamically adjust the speed as a function of the queue length. In this section we investigate the structure of the "optimal" speed scaling algorithm in two ways: (i) we study near-optimal speed scaling rules in the case of both SRPT and PS scheduling; (ii) we study each of these algorithms in both the worst-case model and the stochastic model.

### 2.2.1 Worst-case analysis

There has been significant work studying speed scaling in the worst-case model, focusing on SRPT. A promising algorithm is (SRPT, $P^{-1}(n)$), and there has been a stream upper bounds on its competitive ratio for objective (2.1): for unit-size jobs in [4, 15] and for general jobs with $P(s) = s^\alpha$ in [12, 81]. A major breakthrough was made in [13], which shows the 3-competitiveness of $(SRPT, P^{-1}(n+1))$ for general $P$.

Our contribution to this literature is twofold. First, we tightly characterize the competitive ratio of (SRPT, $P^{-1}(n\beta)$). Specifically, we prove that (SRPT, $P^{-1}(n\beta)$) is exactly 2-competitive under general power functions (see Theorem 2.1 and Corollary 2.1). Second, we prove that no "natural" speed scaling algorithm can be better than 2-competitive. Natural speed scaling algorithms include algorithms which have speeds that grow faster, slower, or proportional to $P^{-1}(n\beta)$, or that use a scheduler that works on exactly one job between arrival/departure events (see Definition 2.1). Thus, the class of natural algorithms includes energy-proportional designs for all schedulers and SRPT scheduling for any $s_n$. We conjecture that this result can be extended to all speed scaling algorithms, which would imply that the competitive ratio of (SRPT, $P^{-1}(n\beta)$) is minimal.

In contrast to this stream of work studying SRPT, there has been no analysis of speed scaling under PS. We prove that (PS, $P^{-1}(n\beta)$) is $O(1)$-competitive for $P(s) = s^\alpha$ with fixed $\alpha$, and in particular is $(4\alpha - 2)$-competitive for typical $\alpha$, i.e., $\alpha \in (1, 3]$. This builds on [34], which studies LAPS, another policy "blind" to job sizes. (LAPS, $P^{-1}(n\beta)$) is also $O(1)$-competitive in this case. For both PS and LAPS the competitive ratio is unbounded for large $\alpha$, which [34] proves holds for all blind policies. But, note that $\alpha \in (1, 3]$ in most computer systems today; thus, asymptotics in $\alpha$ are less important than the performance for small $\alpha$.

The results in this section highlight important insights about fundamental issues in speed scaling design. First, the competitive ratio results highlight that energy-proportional speed scaling ($P(s_n) = n\beta$) is nearly optimal, which provides analytic justification of a common design heuristic, e.g., [16]. Second, note that energy-proportional speed scaling works well for PS and SRPT (and LAPS). This suggests a designer may decouple the choice of a speed scaler from the choice of a scheduler, choices that initially seem very intertwined. Though we have seen this decoupling only for PS, SRPT, and

LAPS, we conjecture that it holds more generally. Third, scheduling seems much less important in the speed scaling model than in the standard constant speed model. For an instance of $\nu$ jobs, PS is $\Omega(\nu^{1/3})$-competitive for mean response time in the constant speed model [96], but is $O(1)$-competitive in the speed scaling model. Again, we conjecture that this holds more generally than for just PS.

**Amortized competitive analysis**

The proofs of the results described above use a technique termed *amortized local competitive analysis* [44, 107]. The technique works as follows.

To show that an algorithm $\mathcal{A}$ is $c$-competitive with an optimal algorithm $OPT$ for a performance metric $z = \int \zeta(t)dt$ it is sufficient to find a *potential function* $\Phi : \mathbb{R} \to \mathbb{R}$ such that, for any instance of the problem:

1. *Boundary condition:* $\Phi = 0$ before the first job is released, and $\Phi \geq 0$ after the last job is finished;
2. *Jump condition:* At any point where $\Phi$ is not differentiable, it does not increase;
3. *Running condition:* When $\Phi$ is differentiable,

$$\zeta^{\mathcal{A}}(t) + \frac{d\Phi}{dt} \leq c\zeta^{O}(t), \tag{2.2}$$

where $\zeta^{\mathcal{A}}(t)$ and $\zeta^{O}(t)$ are the cost $\zeta(t)$ under $\mathcal{A}$ and $OPT$ respectively.

Given these conditions, the competitiveness follows from integrating (2.2), which gives

$$z^{\mathcal{A}} \leq z^{\mathcal{A}} + \Phi(\infty) - \Phi(-\infty) \leq cz^{O}.$$

**SRPT analysis**

We now state and prove our results for SRPT.

**Theorem 2.1.** *For any regular power function $P$, (SRPT, $P^{-1}(n\beta)$) has a competitive ratio of exactly 2.*

The proof of the upper bound is a refinement of the analysis in [13] that accounts more carefully for some boundary cases. It uses the potential function:

$$\Phi(t) = \int_0^\infty \sum_{i=1}^{n[q;t]} \Delta(i)\, dq \tag{2.3}$$

for some non-decreasing $\Delta(\cdot)$ with $\Delta(i) = 0$ for $i \leq 0$, where $n[q;t] = \max(0, n^A[q;t] - n^O[q;t])$ with $n^A[q;t]$ and $n^O[q;t]$ the number of unfinished jobs at time $t$ with remaining size at least $q$ under the scheme under investigation and the optimal (offline) scheme, respectively.

The following technical lemma is the key step of the proof and is proven in Appendix 2.A.

**Lemma 2.1.** *Let $\eta \geq 1$ and $\Phi$ be given by (2.3) with*

$$\Delta(i) = \frac{1+\eta}{\beta} P'\left(P^{-1}\left(i\beta\right)\right). \tag{2.4}$$

*Let $\mathcal{A} = (SRPT, s_n)$ with $s_n \in [P^{-1}(n\beta), P^{-1}(\eta n\beta)]$. Then at points where $\Phi$ is differentiable,*

$$n^{\mathcal{A}} + P(s^{\mathcal{A}})/\beta + \frac{d\Phi}{dt} \leq (1+\eta)(n^O + P(s^O)/\beta). \tag{2.5}$$

Using the above Lemma, we can now prove Theorem 2.1.

*Proof of Theorem 2.1.* To show that the competitive ratio of (SRPT, $P^{-1}(n\beta)$) is at most 2, we show that $\Phi$ given by (2.3) and (2.4) is a valid potential function.

The boundary conditions are satisfied since $\Phi = 0$ when there are no jobs in the system. Also, $\Phi$ is differentiable except when a job arrives or departs. When a job arrives, the change in $n^A[q]$ equals that in $n^O[q]$ for all $q$, and so $\Phi$ is unchanged. When a job is completed, $n[q]$ is unchanged for all $q > 0$, and so $\Phi$ is again unchanged. The running condition is established by Lemma 2.1 with $\eta = 1$.

To prove the lower bound on the competitive ratio, consider periodic unit-work arrivals at rate $\lambda = s_n$ for some $n$. As the number of jobs that arrive grows large, the optimal schedule runs at rate $\lambda$, and maintains a queue of at most one packet (the one in service), giving a cost per job of at most $(1 + P(\lambda)/\beta)/\lambda$. In order to run at speed $\lambda$, the schedule (SRPT, $P^{-1}(n\beta)$) requires $n = P(\lambda)/\beta$ jobs in the queue, giving a cost per job of $(P(\lambda) + P(\lambda))/(\lambda\beta)$. The competitive ratio is thus at least $\frac{2P(\lambda)}{\beta + P(\lambda)}$. As $\lambda$ becomes large, this tends to 2 since a regular $P$ is unbounded. □

Theorem 2.1 can easily be extended to non-negative power functions by applying the same argument as used in [13].

**Corollary 2.1.** *Let $\varepsilon > 0$. For any non-negative and unbounded $\tilde{P}$, there exists a $P$ such that emulating (SRPT, $P^{-1}(n\beta)$) yields a $(2+\varepsilon)$-competitive algorithm.*

This emulation involves avoiding speeds where $P$ is not convex, instead emulating such speeds by switching between a higher and lower speed on the convex hull of $\tilde{P}$.

Corollary 2.1 shows that (SRPT, $P^{-1}(n\beta)$) does not match the performance of the offline optimal. This motivates considering other algorithms; however we now show that no "natural" algorithm can do better.

**Definition 2.1.** *A speed scaling algorithm $\mathcal{A}$ is **natural** if it runs at speed $s_n$ when it has $n$ unfinished jobs, and for convex $P$, one of the following holds:*

*(a) the scheduler is work-conserving and works on a single job between arrival/departure events; or*

*(b) $g(s) + P(s)/\beta$ is convex, for some $g$ with $g(s_n) = n$; or*

*(c) the speeds $s_n$ satisfy $P(s_n) = \omega(n)$; or*

*(d) the speeds $s_n$ satisfy $P(s_n) = o(n)$.*

This fragmented definition seems "unnatural", the class contains most natural contenders for optimality: all algorithms that use the optimal scheduler SRPT, and all whose speeds grow faster than, slower than, or proportional to $P^{-1}(n)$. To be "unnatural", an algorithm must have speeds which increase erratically (or decrease) as $n$ increases.

**Theorem 2.2.** *For any $\varepsilon > 0$ there is a regular power function $P_\varepsilon$ such that any natural algorithm $\mathcal{A}$ on $P_\varepsilon$ has competitive ratio larger than $2 - \varepsilon$.*

This theorem highlights that if an algorithm does have a smaller competitive ratio than (SRPT, $P^{-1}(n\beta)$), it will not use "natural" scheduling or speed scaling. Though the result only applies to natural algorithms, we conjecture that, in fact, it holds for all speed scaling algorithms, and thus the competitive ratio of (SRPT, $P^{-1}(n\beta)$) is minimal.

*Proof.* Consider the case when $P(s) = s^\alpha$, with $\alpha$ yet to be determined. We show that, for large $\alpha$, the competitive ratio is at least $2 - \varepsilon$, by considering two cases: instance $I_{B(\nu)}$ is a *batch arrival* of $\nu$ jobs of size 1 at time 0 with no future arrivals, and instance $I_{R(b,\lambda)}$ is a batch of $b$ jobs at time 0 followed by a long train of *periodic arrivals* of jobs of size 1 at times $k/\lambda$ for $k \in \mathbb{N}$.

Fix an $\varepsilon > 0$ and consider a speed scaling which can attain a competitive ratio of $2 - \varepsilon$ for all instances $I_{R(\cdot,\cdot)}$. For $I_{R(\cdot,\lambda)}$, with large $\lambda$, the optimal algorithm will run at speed exceeding $\lambda$ for a finite time until the occupancy is one. After that, it will run at speed $\lambda$ so that no queue forms. For long trains, this leads to a cost per job of $(1 + P(\lambda)/\beta)/\lambda$.

First, consider a "type (d)" natural $\mathcal{A}$. For sufficiently large $\lambda$, $n > ks_n^\alpha$ for all $s_n \geq \lambda/2$, where $k = 2^{\alpha+2}/\beta$. Between arrivals, at least $1/2$ unit of work must be done at speed at least $\lambda/2$, in order for $\mathcal{A}$ not to fall behind. The cost per unit work is at least $(1/s)(ks^\alpha + s^\alpha/\beta)$, and so the total cost of performing this $1/2$ unit is at least $(k + 1/\beta)\lambda^{\alpha-1}/2^\alpha > 4\lambda^{\alpha-1}/\beta$. For large $\lambda$, this is at least twice the cost per job under the optimal scheme: $(1 + P(\lambda)/\beta)/\lambda < 2\lambda^{\alpha-1}/\beta$.

It remains to consider natural algorithms of types (a)–(c).

Consider a "type (a)" natural $\mathcal{A}$ on the instance $I_{R(n,s_n)}$ for some $n$. It will initially process exactly one job at speed $s_n$, which it will finish at time $1/s_n$. From this time, a new arrival will occur whenever a job completes, and so the algorithm runs at speed $s_n$ with occupancy $n$ until the

last arrival. So, the average cost per job tends to $(n + P(s_n)/\beta)/s_n$ on large instances, leading to a competitive ratio of:

$$1 + \frac{n-1}{P(s_n)/\beta + 1} \leq CR_{periodic} \leq 2 - \varepsilon. \tag{2.6}$$

Consider a "type (b)" natural $\mathcal{A}$. On $I_{R(n,s_n)}$, $\mathcal{A}$ also satisfies (2.6): Let $\bar{s}$ to denote the time-average speed. For all $\phi < 1$, for sufficiently long instances we need $\bar{s} \geq \phi s_n$ to prevent an unbounded queue forming. By Jensen's inequality, the average cost per job satisfies $\bar{z} \geq (g(\bar{s}) + P(\bar{s})/\beta) \geq (g(\phi s_n) + P(\phi s_n)/\beta)$. Since $\phi$ can be arbitrarily close to 1, the cost can be arbitrarily close to $n + P(s_n)/\beta$, implying (2.6).

For a "type (c)" natural $\mathcal{A}$, $P(s_n)/n \to \infty$ for large $n$.

Thus, for types (a)–(c), $\exists n_0$ such that for all $n > n_0$:

$$s_n \geq \hat{s}_n := P^{-1}\left(\frac{n\beta}{1-\varepsilon/2}\right). \tag{2.7}$$

We now show that this condition precludes having a competitive ratio of $2 - \varepsilon$ in the case of batch arrivals, $I_{B(\nu)}$.

For $I_{B(\nu)}$, the optimal strategy is to server one job at a time at some speeds $s_n^*$, giving cost

$$z^O(I_{B(\nu)}) = \sum_{n=1}^{\nu} \frac{n}{s_n^*} + \frac{P(s_n^*)}{\beta s_n^*} = \sum_{n=1}^{\nu} \frac{n^{(\alpha-1)/\alpha}}{\beta^{1/\alpha}} \left[\left(\frac{n\beta}{(s_n^*)^\alpha}\right)^{1/\alpha} + \left(\frac{(s_n^*)^\alpha}{n\beta}\right)^{(\alpha-1)/\alpha}\right].$$

The unique local minimum of $\phi(\cdot) = (\cdot)^{(\alpha-1)/\alpha} + (\cdot)^{-1/\alpha}$ occurs at $1/(\alpha-1)$. This gives a minimum cost of

$$z^O(I_{B(\nu)}) = \frac{\alpha \sum_{n=1}^{\nu} n^{(\alpha-1)/\alpha}}{\beta^{1/\alpha}(\alpha-1)^{(\alpha-1)/\alpha}}$$

for $s_n^* = (n\beta/(\alpha-1))^{1/\alpha}$. More generally, the optimum is

$$\beta n = s_n^* P'(s_n^*) - P(s_n^*). \tag{2.8}$$

Under $\mathcal{A}$, when more than $n-1$ work remains, there must be at least $n$ unfinished jobs. Thus, for $\alpha - 1 > 1 - \varepsilon/2$,

$$z(I_{B(\nu)}) \geq \sum_{n=n_0}^{\nu} \frac{n^{(\alpha-1)/\alpha}}{\beta^{1/\alpha}} \left[\left(\frac{n\beta}{(\hat{s}_n)^\alpha}\right)^{1/\alpha} + \left(\frac{(\hat{s}_n)^\alpha}{n\beta}\right)^{(\alpha-1)/\alpha}\right].$$

since the minimum of $\phi(\cdot)$ subject to (2.7) then occurs at $(\hat{s}_n)^\alpha/(n\beta)$.

Since $(\hat{s}_n)^\alpha/(n\beta) = 1/(1-\varepsilon/2)$, this gives

$$CR_{batch} \geq \left(\frac{\sum_{n=n_0}^{\nu} n^{(\alpha-1)/\alpha}}{\sum_{n=1}^{\nu} n^{(\alpha-1)/\alpha}}\right)\left(\frac{(\alpha-1)^{(\alpha-1)/\alpha}}{\alpha}\right)\left[\left(\frac{1}{1-\varepsilon/2}\right)^{(\alpha-1)/\alpha} + \left(\frac{1}{1-\varepsilon/2}\right)^{-1/\alpha}\right].$$

For any $\varepsilon \in (0,1)$, the product of the last two factors tends to $1 + 1/(1-\varepsilon/2)$ as $\alpha \to \infty$, and hence there is an $\alpha = \alpha(\varepsilon)$ for which their product exceeds $1/(1-\varepsilon/3)+1$. Similarly, for all $\alpha > 1$, there is a sufficiently large $\nu$ that the first factor exceeds $1/(1+\varepsilon/9)$. For this $\alpha$ and $\nu$, $CR_{batch} > 2$.

So, for $P(s) = s^{\alpha(\varepsilon)}$, if the competitive ratio is smaller than $2-\varepsilon$ in the periodic case, it must be larger than 2 in the batch case. □

Theorem 2.2 relies on $P$ being highly convex, as in interference-limited systems [58]. For CMOS systems in which typically $\alpha \in (1,3]$, it is possible to design natural algorithms that can outperform (SRPT, $P^{-1}(n\beta)$).

**PS analysis**

We now state and prove our bound on the competitive ratio of PS.

**Theorem 2.3.** *If $P(s) = s^\alpha$ then (PS, $P^{-1}(n\beta)$) is $\max(4\alpha-2, 2(2-1/\alpha)^\alpha)$-competitive.*

In particular, PS is $(4\alpha-2)$-competitive for $\alpha$ in the typical range of $(1,3]$.

Theorem 2.3 is proven using amortized local competitiveness. Let $\eta \geq 1$, and $\Gamma = (1+\eta)(2\alpha-1)/\beta^{1/\alpha}$. The potential function is then defined as

$$\Phi = \Gamma \sum_{i=1}^{n^A(t)} i^{1-1/\alpha} \max(0, q^A(j_i;t) - q^O(j_i;t)) \tag{2.9}$$

where $q^\pi(j;t)$ is the remaining work on job $j$ at time $t$ under scheme $\pi$, and $\{j_i\}_{i=1}^{n^A(t)}$ is an ordering of the jobs in increasing order of release time: $r(j_1) \leq r(j_2) \leq \cdots \leq r(j_{n^A(t)})$. Note that this is a scaling of the potential function that was used in [34] to analyze LAPS. As a result, to prove Theorem 2.3, we can use the corresponding results in [34] to verify the boundary and jump conditions. All that remains is the running condition, which follows from the technical lemma below. The proof is provided in Appendix 2.B.

**Lemma 2.2.** *Let $\Phi$ be given by (2.9) and $A$ be the discipline (PS, $s_n$) with $s_n \in [(n\beta)^{1/\alpha}, (\eta n\beta)^{1/\alpha}]$. Then under $A$, at points where $\Phi$ is differentiable,*

$$n^A + (s^A)^\alpha/\beta + \frac{d\Phi}{dt} \leq c(n^O + (s^O)^\alpha/\beta) \tag{2.10}$$

*where $c = (1+\eta)\max((2\alpha-1), (2-1/\alpha)^\alpha)$.*

### 2.2.2 Stochastic analysis

We now study optimal dynamic speed scaling in the stochastic setting. In contrast to the worst-case results, in the stochastic setting, it is possible to optimize the algorithm for the expected workload.

In a real application, it is clear that incorporating knowledge about the workload into the design can lead to improved performance. Of course, the drawback is that there is always uncertainty about workload information, either due to time-varying workloads, measurement noise, or simply model inaccuracies. We discuss robustness to these factors in Section 2.4, and in the current section assume that exact workload information is known to the speed scaler and that the model is accurate.

In this setting, there has been a substantial amount of work studying the M/GI/1 PS model [30, 42, 48, 117][3]. This work is in the context of operations management and so focuses on "operating costs" rather than "energy", but the model structure is equivalent. This series of work formulates the determination of the optimal speeds as a stochastic dynamic programming (DP) problem and provides numeric techniques for determining the optimal speeds, as well as proving that the optimal speeds are monotonic in the queue length. The optimal speeds have been characterized as follows [120]. Recall that $\gamma = \rho/\beta^{1/\alpha}$.

**Proposition 2.1.** *Consider an M/GI/1 PS queue with controllable service rates $s_n$. Let $P(s) = s^\alpha$. The optimal dynamic speeds are concave and satisfy the dynamic program given in [120]. For $\alpha = 2$ and any $n \geq 2\gamma$, they satisfy*

$$\gamma + \sqrt{n - 2\gamma} \leq \frac{s_n}{\sqrt{\beta}} \leq \gamma + \sqrt{n} + \min\left(\frac{\gamma}{2n}, \gamma^{1/3}\right). \tag{2.11}$$

*For general $\alpha > 1$, they satisfy*[4]

$$\frac{s_n}{\beta^{1/\alpha}} \leq \left(\frac{1}{\alpha} \min_{\sigma > \gamma} \left(\frac{n + \sigma^\alpha - \gamma^\alpha}{(\sigma - \gamma)} + \frac{\gamma}{(\sigma - \gamma)^2}\right)\right)^{1/(\alpha-1)} \tag{2.12}$$

$$\frac{s_n}{\beta^{1/\alpha}} \geq \left(\frac{n}{\alpha - 1}\right)^{1/\alpha}. \tag{2.13}$$

*Proof.* Bounds (2.11) and (2.12) are shown in [120]. Additionally, the concavity of $s_n$ follows from results in [120]. To prove (2.13), note that when $\rho = 0$ the optimal speeds are those optimal for batch arrivals, which satisfy (2.13) by (2.8). Then, it is straightforward from the DP that $s_n$ increases monotonically with load $\rho$, which gives (2.13). □

Interestingly, the bounds in Proposition 2.1 are tight for large $n$ and have a form similar to the form of the worst-case speeds for SRPT and PS in Theorems 2.1 and 2.3.

In contrast to the large body of work studying the optimal speeds under PS scheduling, there is no work characterizing the optimal speeds under SRPT scheduling. This is not unexpected since the analysis of SRPT in the static speed setting is significantly more involved than that of PS. Thus, instead of analytically determining the optimal speeds for SRPT, we are left to use a heuristic

[3] These actually study the M/M/1 FCFS queue, but since the M/GI/1 PS queue with controllable service rates is a symmetric discipline [72] it has the same occupancy distribution and mean delay as an M/M/1 FCFS queue.

[4] In [120] the range of minimization was misstated as $\sigma > 0$.

approach.

Note that the speeds suggested by the worst-case results for SRPT and PS (Theorems 2.1 and 2.3) are the same, and the optimal speeds for a batch arrival are given by (2.8) for both policies. Motivated by this and the fact that (2.8) matches the asymptotic form of the stochastic results for PS in Proposition 2.1, *we propose to use the optimal PS speeds in the case of SRPT.*

To evaluate the performance of this heuristic, we use simulation experiments (Figure 2.1) that compare the performance of this speed scaling algorithm to the following lower bound.

**Proposition 2.2.** *In a GI/GI/1 queue with* $P(s) = s^\alpha$,

$$z^O \geq \frac{1}{\lambda} \max(\gamma^\alpha, \gamma\alpha(\alpha-1)^{(1/\alpha)-1}).$$

This was proven in [120] in the context of the M/GI/1 PS but the proof can easily be seen to hold more generally.

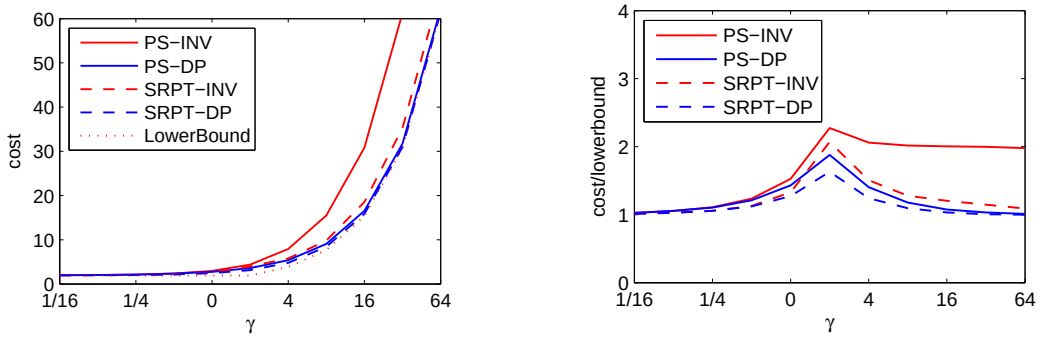
Simulation experiments also allow us to study other interesting topics, such as (i) a comparison of the performance of the worst-case schemes for SRPT and PS with the stochastic schemes and (ii) a comparison of the performance of SRPT and PS in the speed scaling model. In these experiments, the optimal speeds for PS in the stochastic model are found using the numeric algorithm for solving the DP described in [48, 120], and then these speeds are also used for SRPT. Due to limited space, we describe the results from only one of many settings we investigated.

Figure 2.2 shows that the optimal speeds from the DP ("DP") have a similar form to the speeds motivated by the worst-case results, $P^{-1}(n\beta)$ ("INV"), differing by $\gamma$ for high queue occupancies. Figure 2.1 shows how the total cost (2.1) depends on the choice of speeds and scheduler. At low loads, all schemes are indistinguishable. At higher loads, the performance of the PS-INV scheme degrades significantly, but the SRPT-INV scheme maintains fairly good performance. Note though that if $P(s) = s^\alpha$ for $\alpha > 3$ the performance of SRPT-INV degrades significantly too. In contrast, the DP-based schemes benefit significantly from having the slightly higher speeds chosen to optimize (2.1) rather than minimize the competitive ratio. Finally, the SRPT-DP scheme performs nearly optimally, which justifies the heuristic of using the optimal speeds for PS in the case of SRPT[5]. However, the PS-DP scheme performs nearly as well as SRPT-DP. Together, these observations suggest that it is important to optimize the speed scaler, but not necessarily the scheduler.

## 2.3 Gated-static speed scaling

Section 2.2 studied a sophisticated form of speed scaling where the speed can depend on the current occupancy. This scheme can perform (nearly) optimally; however its complexity and overheads may

[5]Note that the peak around $\gamma = 1$ in Fig. 2.1(b) is most likely due to the looseness of the lower bound.




(a) (b)

Figure 2.1: Comparison of SRPT and PS scheduling under both $s_n = P^{-1}(n\beta)$ and speeds optimized for an M/GI/1 PS system, using Pareto(2.2) job sizes and $P(s) = s^2$.

be prohibitive. This is in contrast to the simplest non-trivial form: *gated-static* speed scaling, where $s_n = s_{gs} 1_{n \neq 0}$ for some constant speed $s_{gs}$. This requires minimal hardware to support; e.g., a CMOS chip may have a constant clock speed but AND it with the gating signal to set the speed to 0.

Gated-static speed scaling can be arbitrarily bad in the worst case since jobs can arrive faster than $s_{gs}$. Thus, we study gated-static speed scaling only in the stochastic model, where the constant speed $s_{gs}$ can depend on the load.

We study the gated-static speed scaling under SRPT and PS scheduling. The optimal gated-static speed under PS has been derived in [120], but the optimal speed under SRPT has not been studied previously.

Our results highlight two practical insights. First, we show that gated-static speed scaling can provide nearly the same cost as the optimal dynamic policy in the stochastic model. Thus, the simplest policy can nearly match the performance of the most sophisticated policy. Second, we show that the performance of gated-static under PS and SRPT is not too different, thus scheduling is much less important to optimize than in systems in which the speed is fixed in advance. This reinforces what we observed for dynamic speed scaling.

### 2.3.1 Optimal gated-static speeds

We now derive the optimal speed $s_{gs}$, which minimizes the expected cost of gated-static in the stochastic model under both SRPT and PS. First note that, since the power cost is constant at $P(s_{gs})$ whenever the server is running, the optimal speed is

$$s_{gs} = \arg\min_s \beta \mathbb{E}[T] + \frac{1}{\lambda} P(s) \Pr(N \neq 0). \tag{2.14}$$

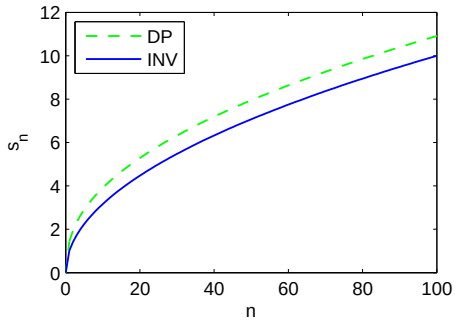



Figure 2.2: Comparison of $s_n = P^{-1}(n\beta)$ with speeds "DP" optimized for an M/GI/1 system with $\gamma = 1$ and $P(s) = s^2$.

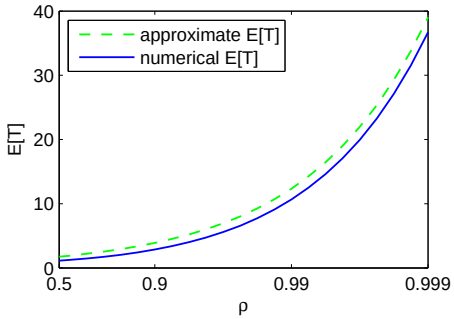



Figure 2.3: Validation of the heavy-traffic approximation (2.19) by simulation using Pareto(3) job sizes with $E[X] = 1$.

In the second term $\Pr(N \neq 0) = \rho/s$, and so multiplying by $\lambda$ and setting the derivative to 0 gives that the optimal gated-static speed satisfies

$$\beta \frac{d\mathbb{E}[N]}{ds} + r\frac{P^*(s)}{s} = 0, \tag{2.15}$$

where $r = \rho/s$ is the utilization and

$$P^*(s) \equiv sP'(s) - P(s). \tag{2.16}$$

Note that if $P$ is convex then $P^*$ is increasing and if $P''$ is bounded away from 0 then $P^*$ is unbounded.

Under PS, $\mathbb{E}[N] = \rho/(s-\rho)$, and so $d\mathbb{E}[N]/ds = \mathbb{E}[N]/(\rho - s)$. By (2.15), the optimal speeds satisfy [120]

$$\beta\mathbb{E}[N] = (1-r)rP^*(s). \tag{2.17}$$

Unfortunately, in the case of SRPT, things are not as easy. For $s = 1$, it is well known, e.g., [75], that

$$\mathbb{E}[T] = \int_{x=0}^{\infty}\int_{t=0}^{x} \frac{dt}{1-\lambda\int_0^t \tau\, dF(\tau)} + \frac{\lambda\int_0^x \tau^2\, dF(\tau) + x^2\bar{F}(x)}{2(1-\lambda\int_0^x \tau dF(\tau))^2}\, dF(x)$$

The complexity of this equation rules out calculating the speeds analytically. So, instead we use simpler forms for $\mathbb{E}[N]$ that are exact in asymptotically heavy or light traffic.

### A heavy-traffic approximation

We state the heavy-traffic results for distributions whose c.c.d.f. $\bar{F}$ has lower and upper Matuszewska indices [20] of $m$ and $M$. Intuitively, $C_1 x^m \lesssim \bar{F}(x) \lesssim C_2 x^M$ as $x \to \infty$ for some $C_1$, $C_2$. So, the Matuszewska index can be thought of as a "moment index." Further, let $G(x) = \int_0^x tf(t)\, dt/\mathbb{E}[X]$

be the fraction of work coming from jobs of size at most $x$. The following was proven in [85].

**Proposition 2.3** ([85]). *For an $M/GI/1$ under SRPT with speed 1, $\mathbb{E}[N] = \Theta(H(\rho))$ as $\rho \to 1$, where*

$$H(\rho) = \begin{cases} E[X^2]/((1-\rho)G^{-1}(\rho)) & \text{if } M < -2 \\ E[X]\log(1/(1-\rho)) & \text{if } m > -2. \end{cases} \tag{2.18}$$

Proposition 2.3 motivates the heavy-traffic approximation below for the case when the speed is 1:

$$\mathbb{E}[N] \approx CH(\rho) \tag{2.19}$$

where $C$ is a constant dependent on the job size distribution. For job sizes which are Pareto($a$) (or more generally, regularly varying [20]) with $a > 2$, it is known that $C = (\pi/(1-a))/(2\sin(\pi/(1-a)))$ [85]. Figure 2.3 shows that in this case, the heavy-traffic results are accurate even for quite low loads.

Given approximation (2.19), we can now return to equation (2.15) and calculate the optimal speed for gated-static SRPT. Define $h(r) = (G^{-1})'(r)/G^{-1}(r)$.

**Theorem 2.4.** *Suppose approximation (2.19) holds with equality.*

*(i) If $M < -2$, then for the optimal gated-static speed,*

$$\beta\mathbb{E}[N]\left(\frac{2-r}{1-r} - rh(r)\right) = rP^*(s). \tag{2.20a}$$

*(ii) If $m > -2$, then for the optimal gated-static speed,*

$$\beta\mathbb{E}[N]\left(\frac{1}{(1-r)\log(1/(1-r))}\right) = P^*(s). \tag{2.20b}$$

*Proof.* For brevity, we only prove the second claim.

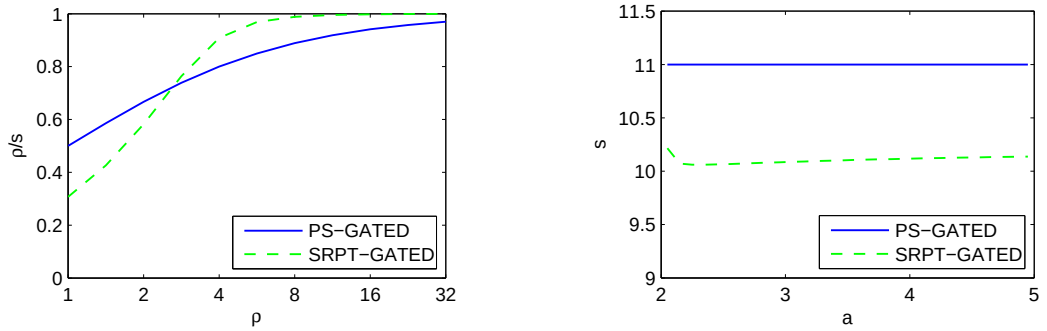
If $m > -2$, then there is a $C' = CE[X]$ such that

$$\mathbb{E}[N] = \frac{C'}{s}\log\left(\frac{1}{1-\rho/s}\right). \tag{2.21}$$

for speed $s$. Now

$$\begin{aligned} \frac{d\mathbb{E}[N]}{ds} &= -\frac{C'}{s^2}\log\left(\frac{1}{1-\rho/s}\right) - \frac{C'\rho}{s^2(s-\rho)} \\ &= -\frac{\mathbb{E}[N]}{s}\left(1 + \frac{\rho}{s}\frac{1}{(1-\rho/s)\log(1/(1-\rho/s))}\right), \end{aligned}$$

and the factor in brackets is dominated by its second term in heavy traffic. Substituting this into (2.15) gives the result. □




(a) (b)

Figure 2.4: Comparison for gated-static: PS using (2.17) and SRPT using (2.22), with $P(s) = s^2$. (a) Utilization given Pareto(2.2) job sizes. (b) Dependence of speed on the job size distribution, for Pareto($a$).

To evaluate the speeds derived for heavy-traffic, Figure 2.4(b) illustrates the gated-static speeds derived for SRPT and PS, for $P(s) = s^2$ and $\rho = 10$ and varying job size distribution. This suggests that the SRPT speeds are nearly independent of the job size distribution. (Note that the vertical axis does not start from 0.) Moreover, the speeds of SRPT and PS differ significantly in this setting since the speeds under SRPT are approximately minimal (the speeds must be larger than $\gamma$), while the PS speeds are $\gamma + 1$.

**Beyond heavy-traffic regime**

Let us next briefly consider the light-traffic regime. As $\rho \to 0$, there is seldom more than one job in the system, and SRPT and PS have nearly indistinguishable $\mathbb{E}[N]$. So, in this case, it is appropriate to use speeds given by (2.17).

Given the light-traffic and heavy-traffic approximations we have just described, it remains to decide the speed in the intermediate regime. We propose setting

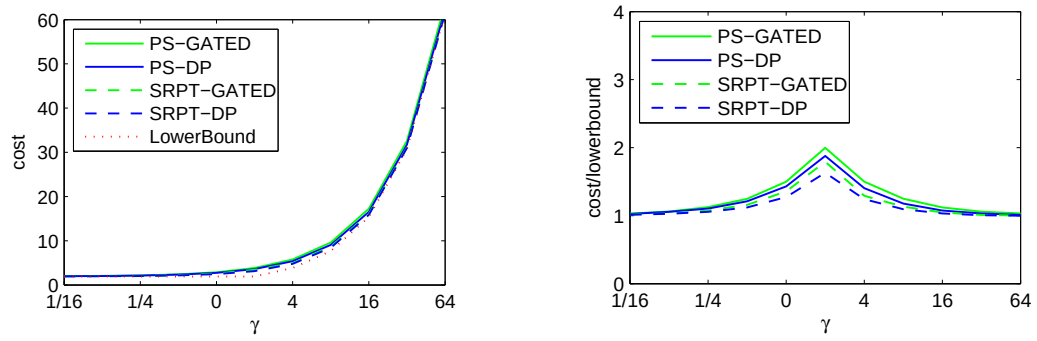
$$s_{gs}^{SRPT} = \min(s_{gs}^{PS}, s_{gs}^{SRPT(HT)}), \tag{2.22}$$

where $s_{gs}^{PS}$ satisfies (2.17), and $s_{gs}^{SRPT(HT)}$ is given by (2.20) with $\mathbb{E}[N]$ estimated by (2.19).

To see why (2.22) is reasonable, we first show that (2.20) often tends to the optimal speed as $\rho \to 0$.

**Proposition 2.4.** *If $m > -2$ or both $M < -2$ and arbitrarily small jobs are possible (i.e., for all $x > 0$ there is a $y \in [0, x]$ with $F(y) > 0$), then (2.20) produces the optimal scaling as $\rho \to 0$.*

*Proof.* For $\rho \to 0$, also $r \to 0$, and $\mathbb{E}[N]/r \to 1$. By L'Hospital's rule $(1-r)\log(1/(1-r))/r \sim 1$, and (2.20b) gives $\beta = P^*(s)$. If arbitrarily small jobs are possible, then $G^{-1}(0) = 0$, and $rh(r) \to 1$ by L'Hospital's rule, whence (2.20a) also becomes $\beta = P^*(s)$.




(a) (b)

Figure 2.5: Comparison of PS and SRPT with gated-static speeds (2.17) and (2.22), versus the dynamic speeds optimal for an M/GI/1 PS. Job sizes are distributed as Pareto(2.2) and $P(s) = s^2$.

From (2.8), this is the optimal speed at which to server a batch of a single job. Since, as $\rho \to 0$, the system almost certainly has a single job when it is non-empty, this is an appropriate speed. □

Although (2.20) tends to the optimal speeds, (2.19) over estimates $\mathbb{E}[N]$ for small $\rho$ and so $s_{gs}^{SRPT(HT)}$ is higher than optimal for small loads. Conversely, for a given speed, the delay is less under SRPT than PS, and so the optimal speed under SRPT will be lower than that under PS. Hence $s_{gs}^{SRPT(HT)} < s_{gs}^{PS}$ in the large $\rho$ regime where the former becomes accurate. Thus, the min operation in (2.22) selects the appropriate form in each regime.

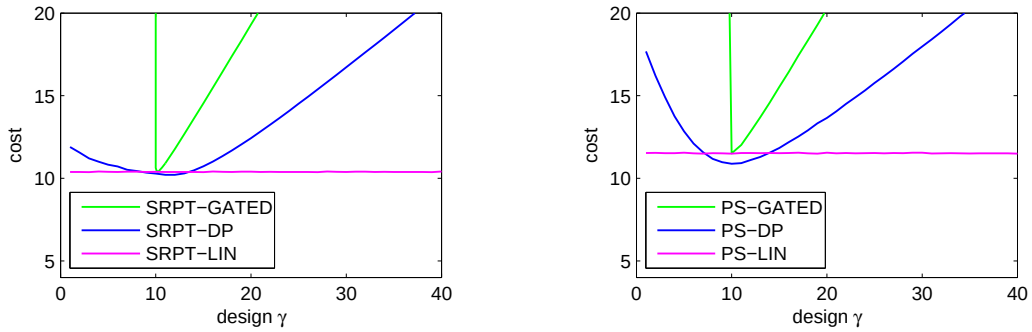
### 2.3.2 Gated-static vs. dynamic speed scaling

Now that we have derived the optimal gated-static speeds, we can contrast the performance of gated-static with that of dynamic speed scaling. This is a comparison of the most and least sophisticated forms of speed scaling.

As Figure 2.5 shows, the performance (in terms of mean delay plus mean energy) of a well-tuned gated-static system is almost indistinguishable from that of the optimal dynamic speeds. Moreover, there is little difference between the cost under PS-GATED and SRPT-GATED, again highlighting that the importance of scheduling in the speed scaling model is considerably less than in standard queueing models.

In addition to observing numerically that the gated-static schemes are near optimal, it is possible to provide some analytic support for this fact as well. In [120] it was proven that PS-GATED is within a factor of 2 of PS-DP when $P(s) = s^2$. Combining this result with the competitive ratio results, we have

**Corollary 2.2.** *Consider $P(s) = s^2$. The optimal PS and SRPT gated-static designs are $O(1)$-competitive in an M/GI/1 queue with load $\rho$.*




(a) SRPT (b) PS

Figure 2.6: Effect of misestimating $\gamma$ under PS and SRPT: cost when $\gamma = 10$, but $s_n$ are optimal for a different "design $\gamma$". Pareto(2.2) job sizes; $P(s) = s^2$.

*Proof.* Let $\pi \in \{PS, SRPT\}$ and $s^{\pi}_{gs}$ be the optimal gated-static speed for $\pi$ and $s^{DP}_n$ be the optimal speeds, which solve the DP for the M/GI/1 PS queue. Then

$$z^{(\pi, s^{\pi}_{gs})} \leq z^{(PS, s^{PS}_{gs})} \leq 2z^{(PS, s^{DP}_n)} \leq 2z^{(PS, P^{-1}(n\beta))} \leq 12z^{O}.$$

The last three steps follow from [120], the optimality of DP for PS in M/GI/1, and Theorem 2.3. □

## 2.4 Robustness and speed scaling

Section 2.3 shows that near-optimal performance can be obtained using the simplest form of speed scaling — running at a static speed when not idle. Why then do CPU manufacturers design chips with multiple speeds? The reason is that the optimal gated-static design depends intimately on the load $\rho$. This cannot be known exactly in advance, especially since workloads typically vary over time. So, an important property of a speed scaling design is *robustness* to uncertainty in the workload, $\rho$ and $F$, and to model inaccuracies.

Figure 2.6 illustrates that if a gated-static design is used, performance degrades dramatically when $\rho$ is mispredicted. If the static speed is chosen and the load is lower than expected, excess energy will be used. Underestimating the load is even worse; if the system has static speed $s$ and $\rho \geq s$ then the cost is unbounded.

In contrast, Figure 2.6 illustrates simulation experiments which show that dynamic speed scaling (SRPT-DP) is significantly more robust to misprediction of the workload. In fact, we can prove this analytically by providing worst-case guarantees for the SRPT-DP and PS-DP. Let $s^{DP}_n$ denote the speeds used for SRPT-DP and PS-DP. Note that the corollary below is distinctive in that it provides worst-case guarantees for a stochastic control policy.

**Corollary 2.3.** *Consider $P(s) = s^\alpha$ with[6] $\alpha \in (1, 2]$ and algorithm $\mathcal{A}$ which chooses speeds $s_n^{DP}$ optimal for PS scheduling in an M/GI/1 queue with load $\rho$. If $\mathcal{A}$ uses either PS or SRPT scheduling, then $\mathcal{A}$ is $O(1)$-competitive in the worst-case model.*

*Proof.* The proof applies Lemmas 2.1 and 2.2 from the worst-case model to the speeds from the stochastic model.

By Proposition 2.1, $s_n \geq (n\beta/(\alpha-1))^{1/\alpha}$. Since $\alpha < 2$, this implies $s_n \geq P^{-1}(n\beta)$. Further, (2.12) implies that $s_n^{DP} = O(n^{1/\alpha})$ for any fixed $\rho$ and $\beta$ and is bounded for finite $n$.

Hence the speeds $s_n^{DP}$ are of the form given in Lemmas 2.1 and 2.2 for some finite $\eta$ (dependent on $\pi$ and the constant $\rho$), from which it follows that $\mathcal{A}$ is constant competitive. □

For $\alpha = 2$, Proposition 2.1 implies $s_n^{DP} \leq (2\gamma+1)P^{-1}(n\beta)$, whence (SRPT, $s_n^{DP}$) is $(2\gamma+2)$-competitive.

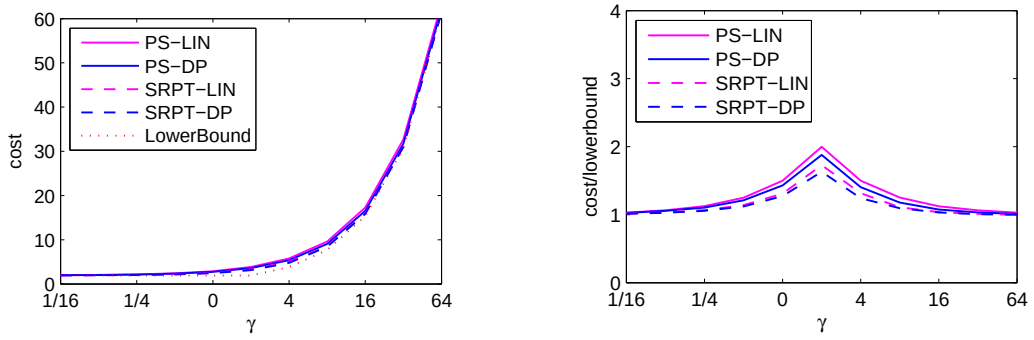
Corollary 2.3 highlights that $s_n^{DP}$ designed for a given $\rho$ leads to a speed scaler that is "robust". However, the cost still degrades significantly when $\rho$ is mispredicted badly (as shown in Figure 2.6).

We now consider a different form of robustness: If the arrivals are known to be well approximated by a Poisson process, but $\rho$ is unknown, is it possible to choose speeds that are close to optimal for all $\rho$? It was shown in [120] that using "linear" speeds, $s_n = n\sqrt{\beta}$, gives near-optimal performance when $P(s) = s^2$ and PS scheduling is used. This scheme ("LIN") performs much better than using $s_n = P^{-1}(n\beta)$, despite the fact that it also uses no knowledge of the workload. Given the decoupling of scheduling and speed scaling suggested by the results in Section 2.2, this motivates using the same linear speed scaling for SRPT. Figure 2.7 illustrates that this linear speed scaling provides near-optimal performance under SRPT too. The robustness of this speed scaling is illustrated in Figure 2.6. However, despite being more robust in the sense of this paragraph, the linear scaling is not robust to model inaccuracies. Specifically, it is not $O(1)$-competitive in general, nor even for the case of batch arrivals.

## 2.5 Fairness and speed scaling

To this point we have seen that speed scaling has many benefits; however we show in this section that dynamic speed scaling has an undesirable consequence — magnifying unfairness. Fairness is an important concern for system design in many applications, and the importance of fairness when considering energy efficiency was recently raised in [109]. However, unfairness under speed scaling designs has not previously been identified. In retrospect though, it is not a surprising byproduct of speed scaling: If there is some job type that is always served when the queue length is long/short it will receive better/worse performance than it would have in a system with a static speed. To see

[6] This is proven in [121] for $\alpha \in (1, \infty)$.




(a) (b) c

Figure 2.7: Comparison of PS and SRPT with linear speeds, $s_n = n\sqrt{\beta}$, and with dynamic speeds optimal for PS. Job sizes are Pareto(2.2) and $P(s) = s^2$.

that this magnifies unfairness, rather than being independent of other biases, note that the scheduler has greatest flexibility to select which job to serve when the queue is long, and so jobs served at that time are likely to be those that already get better service.

In this section, we prove that this service rate differential can lead to unfairness in a rigorous sense under SRPT and non-preemptive policies such as first come first serve (FCFS, which serves jobs in order of arrival). However, under PS, speed scaling does not lead to unfairness.

### 2.5.1 Defining fairness

The fairness of scheduling policies has recently received a lot of attention in computer systems modeling, which has led to a variety of fairness measures, e.g., [11, 105, 122], and the analysis of nearly all common scheduling policies, e.g., [74, 103, 122]. Refer to the survey [119] for more details.

Here, we compare fairness not between individual jobs, but between classes of jobs, where a class consists of all jobs of a given size. Since we focus on delay, we compare $\mathbb{E}[T(x)]$ across $x$. For this purpose, fairness when $s = 1$ has been defined in prior work as follows [119]:

**Definition 2.2.** *A policy $\pi$ is fair if for all $x$*

$$\frac{\mathbb{E}[T^{\pi}(x)]}{x} \leq \frac{\mathbb{E}[T^{PS}(x)]}{x}.$$

This metric is motivated by the fact that (i) PS is intuitively fair since it shares the server evenly among all jobs at all times; (ii) for $s = 1$, the slowdown ("stretch") of PS is constant, i.e., $\mathbb{E}[T(x)]/x = 1/(1-\rho)$; (iii) $\mathbb{E}[T(x)] = \Theta(x)$ [59], so normalizing by $x$ when comparing the performance of different job sizes is appropriate. Additional support is provided by the fact that $\min_{\pi} \max_{x} \mathbb{E}[T^{\pi}(x)]/x = 1/(1-\rho)$ [122].

Using this definition, it is interesting to note that the class of large jobs is always treated fairly under all work-conserving policies, i.e., $\lim_{x\to\infty} \mathbb{E}[T(x)]/x \leq 1/(1-\rho)$ [59] — even under policies such as SRPT that seem biased against large jobs. In contrast, all non-preemptive policies, e.g., FCFS have been shown to be unfair to small jobs [122].

The foregoing applies when $s = 1$. The following proposition shows that PS still maintains a constant slowdown in the speed scaling environment, and so Definition 2.2 is still a natural notion of fairness.

**Proposition 2.5.** *Consider an M/GI/1 queue with a symmetric scheduling discipline, e.g., PS with controllable service rates. Then,* $\mathbb{E}[T(x)] = x\left(\mathbb{E}[T]/\mathbb{E}[X]\right)$.

The proof follows from using Little's law for jobs with size in $[x, x+\epsilon]$ and is omitted here.

### 2.5.2 Speed scaling magnifies unfairness

Now that we have a natural criterion for fairness, we prove that speed scaling creates/magnifies unfairness under SRPT and non-preemptive policies such as FCFS.

#### SRPT

We first prove that SRPT treats the largest jobs unfairly in a speed scaling system. Recall that the largest jobs are always treated fairly in the case of a static speed.

Let $\bar{s}^{\pi}$ be the time-average speed under policy $\pi$, and let $\pi+1$ denote running policy $\pi$ on a system with a permanent customer in addition to the stochastic load (e.g., $\bar{s}^{PS+1}$).

**Theorem 2.5.** *Consider a GI/GI/1 queue with controllable service rates and unbounded inter-arrival times. Let* $s_n^{SRPT} \leq s_n^{PS}$ *be weakly monotone increasing and satisfy* $\bar{s}^{PS+1} > \rho$ *and* $\bar{s}^{SRPT+1} > \rho$.[7] *Then*

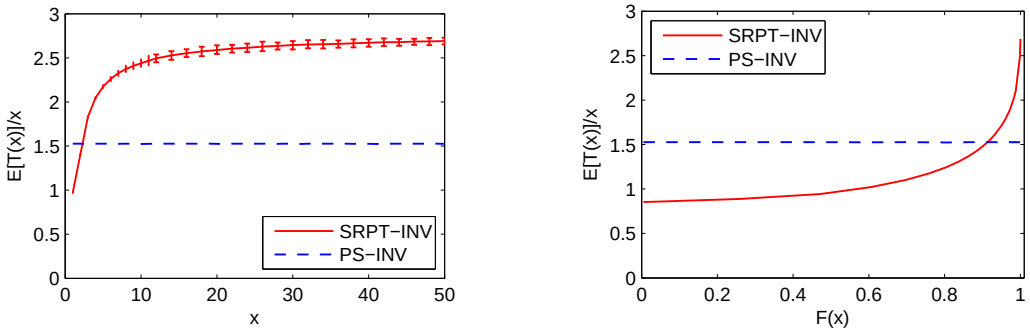
$$\lim_{x\to\infty} \frac{T^{PS}(x)}{x} <_{a.s.} \lim_{x\to\infty} \frac{T^{SRPT}(x)}{x}.$$

The intuition behind Theorem 2.5 is the following. An infinitely sized job under SRPT will receive almost all of its service while the system is empty of smaller jobs. Thus it receives service during the idle periods of the rest of the system. Further, if $s_n^{SRPT} \leq s_n^{PS}$ then the busy periods will be longer under SRPT and so the slowdown of the largest job will be strictly greater under SRPT. This intuition also provides an outline of the proof.

*Proof.* By Lemma 2.6 in Appendix 2.C, $T^{\pi}(x)/x \to 1/(\bar{s}^{\pi+1} - \rho)$ a.s. in each case.

Lemma 2.9 completes the proof by showing $\bar{s}^{PS+1} > \bar{s}^{SRPT+1}$. It considers the average speed between renewal instants in which both queues are empty, which it maps to renewal periods. It then

[7] Note that the conditions $\bar{s}^{PS+1} > \rho$ and $\bar{s}^{SRPT+1} > \rho$ are equivalent to the stability conditions for $s_n^{SRPT}$ and $s_n^{PS}$.




(a) vs job size (b) vs CDF of job size

Figure 2.8: Slowdown of large jobs under PS and SRPT under Pareto(2.2) job sizes, $\gamma = 1$, $s_n = P^{-1}(n)$, and $P(s) = s^2$. Note the fairness of PS.

uses Lemma 2.7, which shows that a busy period is longer under SRPT than PS, to show that less work is done on the permanent customer in the renewal period under SRPT than under PS. □

Figure 2.8 shows that unfairness under SRPT can be considerable, with large jobs suffering a significant increase in slowdown as compared to PS. However, in this case only around 10% of the jobs are worse off than under PS. Note that this setting has a moderate load, which means that SRPT with static speeds would be fair to all job sizes. Figure 2.8 was generated by running a simulation to steady state and then injecting a job of size $x$ into the system and measuring its response time. This was repeated until the 90% confidence intervals (shown on Figure 2.8(a) for SRPT) were tight around the estimate.

Theorem 2.5 proves that SRPT cannot use dynamic speeds and provide fairness to large jobs; however, by using gated-static speed scaling SRPT can provide fairness, e.g., [122]. Further, as Figure 2.5 illustrates, gated-static speed scaling provides nearly optimal cost. So, it is possible to be fair and near-optimal using SRPT scheduling but, to be fair, robustness must be sacrificed.

### Non-preemptive policies

The magnification of unfairness by speed scaling also occurs for all non-preemptive policies.

In the static speed setting, all non-preemptive policies are unfair to small jobs [122] since the response time must include at least the residual of the job size distribution if the server is busy, i.e.,

$$\mathbb{E}[T(x)]/x \geq 1 + \rho\mathbb{E}[X^2]/(2\mathbb{E}[X]x),$$

which grows unboundedly as $x \to 0$. However, if we condition on the arrival of a job to an empty system (i.e., the work in system at arrival $W = 0$), then non-preemptive policies are "fair", in the sense that the slowdown is constant: $T(x|W = 0)/x = 1$. Speed scaling magnifies unfairness under

non-preemptive policies in the following sense: $T(x|W=0)/x$ can now differ dramatically across job sizes.

**Proposition 2.6.** *Consider a non-preemptive GI/GI/1 speed scaling queue with mean inter-arrival time $1/\lambda$ and speeds $s_n$ monotonically approaching $s_\infty \in (0,\infty]$ as $n \to \infty$. Then, with probability 1,*

$$\lim_{x\to 0} \frac{T(x|W=0)}{x} = \frac{1}{s_1} \quad \textit{and} \quad \lim_{x\to\infty} \frac{T(x|W=0)}{x} = \frac{1}{s_\infty}.$$

The intuition behind this result is that small jobs receive their whole service while alone in the system; whereas large jobs have a large queue build up behind them, and therefore get served at a faster speed. Thus, the service rate of large and small jobs differs, magnifying the unfairness of non-preemptive policies.

*Proof.* First, the limit as $x \to 0$ follows immediately from noting that as $x$ shrinks the probability of another arrival before completion goes to 0.

To prove the limit as $x \to \infty$, let $\tilde{A}(x)$ be such that

$$\sum_{i=0}^{\tilde{A}(x)-1} \frac{s_i}{\lambda} < x \leq \sum_{i=0}^{\tilde{A}(x)} \frac{s_i}{\lambda}.$$

and let $\epsilon > 0$ be arbitrary. This $\tilde{A}(x)$ can be thought of as the number of arrivals before $x$ work is completed if jobs arrived periodically with inter-arrival time $1/\lambda$.

Since speeds are non-decreasing, the time to finish the job can be bounded above by the time to reach speed $s_i$ plus the time it would take to finish the whole job at speed $s_i$. Further, we can use the law of large numbers to bound the time to reach speed $s_i$ as $x \to \infty$. This gives

$$Pr\left(\frac{T(x|W=0)}{x} < \frac{1}{s_{\sqrt{\tilde{A}(x)}}} + \frac{\sqrt{\tilde{A}(x)}}{x}\frac{1+\epsilon}{\lambda}\right) \to 1 \text{ w.p.1.} \tag{2.23}$$

Since $\{s_i\}$ are non-decreasing and $\tilde{A}(x) = \Theta(x)$, it follows that the right hand side inside the brackets approaches $1/s_\infty$ as $x \to \infty$.

Conversely, a lower bound on the time to finish the job is given by the time to finish it at maximum speed:

$$Pr\left(\frac{T(x|W=0)}{x} \geq \frac{1}{s_\infty}\right) = 1 \text{ w.p.1.} \tag{2.24}$$

Together, (2.23) and (2.24) establish the result. □

In general, speed scaling based on the occupancy $n$ may magnify unfairness in any policy for which $n(t)$ is correlated with the size of the job(s) being processed at time $t$. Note that gated-static scaling does not magnify unfairness, regardless of the scheduling discipline, since all jobs are processed at the same speed.

## 2.6 Concluding remarks

This chapter has studied several fundamental questions about the design of speed scaling algorithms. The focus has been on understanding the structure of the optimal algorithm, the interaction between speed scaling and scheduling, and the impact of the sophistication of the speed scaler. This has led to a number of new insights, which are summarized in the introduction.

The analytic approach of this chapter is distinctive in that it considers both worst-case and stochastic models. This combination of techniques is fundamental in obtaining two of the main results of the work: Corollary 2.3 providing worst-case guarantees for policies designed in the stochastic model, and Theorem 2.5 identifying unfairness in expected performance under dynamic speed scaling with SRPT. Further, the combination of stochastic and worst-case analysis adds support to many of the other insights of the chapter, e.g., the decoupling of scheduling and speed scaling.

These results suggest many interesting topics. Foremost, it will be interesting to see if the lower bound of 2-competitive for natural speed scaling algorithms extends to all algorithms. It is also important to understand the range of applicability of the insights that speed scaling can be decoupled from scheduling with little performance loss, and that scheduling is less important when energy is added to the objective. Further, the implications for fairness were only touched on briefly. Finally, it is important to address all of the issues studied here in the context of other performance objectives, e.g., when temperature is considered or when more general combinations of energy and response time are considered.

## Appendix 2.A Running condition for SRPT

The proof of Lemma 2.1 uses the following lemmas, which parallel those in [13]. Let $n^A(\cdot)$ and $n^O(\cdot)$ be arbitrary unfinished work profiles, $n^A = n^A(0)$, $n^O = n^O(0)$, and let $s^A$ and $s^O$ be arbitrary non-negative speeds, with $s^O = 0$ if $n^O = 0$.

**Lemma 2.3.** *For any non-decreasing $\Delta$ with $\Delta(i) = 0$ for $i \leq 0$, if $n^O < n^A$ then, under SRPT, where $\Phi$ is differentiable either*

$$\textit{both} \quad \tfrac{d}{dt}\Phi \leq \Delta(n^A - n^O + 1)(-s^A + s^O) \tag{2.25a}$$

$$\textit{and} \quad n^O \geq 1, \tag{2.25b}$$

$$\textit{or} \quad \tfrac{d}{dt}\Phi \leq \Delta(n^A - n^O)(-s^A + s^O). \tag{2.25c}$$

*Proof.* Consider an interval $I = [t, t + dt]$ sufficiently small that no arrivals or departures occur. Let $\Phi(t + dt) - \Phi(t) = d\Phi^A + d\Phi^O$, where $d\Phi^A$ reflects the change in $n^A$ and $d\Phi^O$ reflects the change due to OPT. On $I$, $n^x[q]$ decreases by 1 for $q \in [q^x - s^x\, dt, q^x]$, for $x = A, OPT$. Then $A$ will remove

a term from the sum in (2.3), and $OPT$ may add an additional term. Let $q^A$ ($q^O$) be the remaining work of the job being processed by algorithm $A$ ($OPT$). If $q^A \neq q^O$, these intervals do not overlap, and so

$$d\Phi^A = -\Delta(n^A[q^A] - n^O[q^A])s^A\,dt \tag{2.26a}$$

$$d\Phi^O \leq \Delta(n^A[q^O] - (n^O[q^O] - 1))s^O\,dt. \tag{2.26b}$$

The result follows from one of the following cases, divided by $dt$. The improvement from [13] comes from handling the boundary case $n^O = 0$ more carefully.

$q^A < q^O$ The second term in (2.26a) becomes $n^O[q^A] = n^O[q^O] = n^O$, whence $d\Phi^A = -\Delta(n^A - n^O)s^A\,dt$. Since $q^A < q^O$ implies $n^A[q^O] \leq n^A[q^A] - 1$, and $\Delta$ is non-decreasing, $\Delta(n^A[q^O] - (n^O[q^O] - 1)) \leq \Delta(n^A[q^A] - n^O[q^O])$. Thus $d\Phi^A + d\Phi^O \leq \Delta(n^A - n^O)(-s^A + s^O)\,dt$.

$q^A = q^O$ If $s^A \geq s^O$ then one term is removed from the sum in (2.3) for $q \in [q^A - s^A\,dt, q^A - s^O\,dt]$, which gives $\Phi(t + dt) - \Phi(t) = \Delta(n^A - n^O)(-s^A + s^O)\,dt$.

If $s^O > s^A$, then one term is added for $q \in [q^A - s^O\,dt, q^A - s^A\,dt]$, whence $\Phi(t + dt) - \Phi(t) = \Delta(n^A - n^O + 1)(-s^A + s^O)\,dt$. As $s^O > s^A \geq 0$, $n^O \neq 0$, whence $n^O \geq 1$.

$q^A > q^O$ If $n^O = 0$ then, $n^O[q^A] = 0 = n^O$ whence $d\Phi^A \leq -\Delta(n^A - n^O)s^A\,dt$, and $s^O = 0$ whence $d\Phi^O = 0 = 2\Delta(n^A - n^O)s^O\,dt$. This implies (2.25c).

If $n^O > 1$ then $q^A > q^O$ implies $n^O[q^A] \leq n^O[q^O] - 1$. Since $\Delta$ is non-decreasing, (2.26a) becomes $d\Phi^A \leq -\Delta(n^A - n^O + 1)s^A\,dt$. Since $q^A > q^O$, $n^A[q^O] = n^A[q^A] = n^A$, and (2.26b) becomes $d\Phi^O \leq \Delta(n^A - n^O + 1)s^O\,dt$. This implies (2.25a) and (2.25b).

□

This differs from the corresponding result in [13] in condition (2.25b), which ensures that the argument of $\Delta$ in (2.25) is always at most $n^A$ and gives the following.

**Lemma 2.4.** *Consider a regular power function, $P$, and let $\Delta(i)$ be given by (2.4) for $i > 0$. If $n^O < n^A$ and $n^A \leq P(s^A)/\beta$ then (2.25) implies*

$$\frac{d\Phi}{dt} \leq (1 + \eta)(P(s^O)/\beta - n^A + n^O).$$

*Proof.* Since $P$ is regular, $\Delta$ is non-decreasing. Now, consider two cases.

If (2.25a) and (2.25b) holds, then let $\Psi(s) = P(s)/\beta$ and set $i = n^A - n^O + 1$ in Lemma 2.5

below to give

$$\begin{aligned}\frac{d\Phi}{dt} &\le \Delta(n^A - n^O + 1)(-s^A + s^O)\\ &= (1+\eta)\Psi'(\Psi^{-1}(n^A - n^O + 1))(-s^A + s^O)\\ &\le (1+\eta)\big(-s^A + \Psi^{-1}(n^A - n^O + 1)\big)\Psi'(\Psi^{-1}(n^A - n^O + 1)) + (1+\eta)(\Psi(s^O) - n^A + n^O - 1)\\ &\le (1+\eta)(\Psi(s^O) - n^A + n^O)\end{aligned}$$

where the last inequality follows from

$$n^O \ge 1 \quad \Rightarrow \quad s^A \ge P^{-1}(n^A\beta) \ge \Psi^{-1}(n^A - n^O + 1). \tag{2.27}$$

Otherwise (2.25c) holds. Since $s^A \ge P^{-1}(n^A\beta) \ge \Psi^{-1}(n^A - n^O)$, the above manipulations go through again, with $i = n^A - n^O$ in Lemma 2.5. □

Next, we need the following result, Lemma 3.1 of [13].

**Lemma 2.5.** *[13] Let $\Psi$ be a strictly increasing, strictly convex, differentiable function. Let $i, s^A, s^O \ge 0$ be real. Then*

$$\Psi'(\Psi^{-1}(i))(-s^A + s^O) \le (-s^A + \Psi^{-1}(i))\Psi'(\Psi^{-1}(i)) + \Psi(s^O) - i.$$

We can now prove Lemma 2.1.

*Proof of Lemma 2.1.* When $n^A = 0$, (2.5) holds trivially. Consider now three cases when $n^A \ge 1$:

If $n^O > n^A$, then $d\Phi^O = 0$, since there is a $dt > 0$ such that $n^O[q] > n^A[q]$ for $q \in [q^O - s^O\,dt, q^O]$, which implies that $n^O[q] - n^A[q] \le 0$ for all times in $[t, t+dt]$. Since $d\Phi^A \le 0$ on any interval, $d\Phi \le 0$. Thus (2.5) holds, since $P(s^A)/\beta \le \eta n^A$.

Consider next $n^O < n^A$. Since the optimal scheme runs at zero speed when it is empty, $s^O = 0$ if $n^O = 0$, and so Lemma 2.3 applies. Then by Lemma 2.4, $d\Phi/dt \le (1+\eta)(P(s^O)/\beta - n^A + n^O)$, whence

$$n^A + \frac{P(s^A)}{\beta} + \frac{d}{dt}\Phi \le n^A + \eta n^A + (1+\eta)(\frac{P(s^O)}{\beta} - n^A + n^O) = (1+\eta)(n^O + P(s^O)/\beta).$$

Finally, if $n^O = n^A$, then either $d\Phi \le 0$ or (2.25) holds:

1. If $q^A < q^O$, then $n^A[q] - n^O[q]$ becomes negative for $q \in [q^A - s^A\,dt, q^A]$ (whence $n[q] = \max(0, n^A[q] - n^O[q])$ remains 0), and remains negative for $q \in [q^O - s^O\,dt, q^O]$. Hence $n[q]$ is unchanged, and $d\Phi = 0$.

2. If $q^A = q^O$, consider two cases. (i) If $s^A \geq s^O$ then $n^A[q] - n^O[q]$ becomes negative for $q \in [q^A - s^A\,dt, q^A - s^O\,dt]$ and remains zero for $q \in [q^A - s^O\,dt, q^A]$, whence $n[q]$ again remains unchanged. (ii) Otherwise, $n[q]$ increases by 1 for $q \in [q^A - s^O\,dt, q^A - s^A\,dt]$, and $d\Phi = \Delta(n^A - n^O + 1)(-s^A + s^O)\,dt$. Again, $n^O \geq 1$ since $s^O > s^A \geq 0$, and the optimal is idle when $n^O = 0$.

3. The case $q^A > q^O$ is identical to the case in the proof of Lemma 2.3, and (2.25) holds.

Again, if (2.25) holds, then (2.5) holds. If instead $d\Phi \leq 0$, then the left hand side of (2.5) is at most $(1+\eta)n^A$, which is less than the first term on the right hand side. □

## Appendix 2.B Running condition for PS

*Proof of Lemma 2.2.* First note that if $n^A = 0$ then the LHS of (2.10) is 0, and the inequality holds. Henceforth, consider the case $n^A \geq 1$.

The rate of change of $\Phi$ caused by running OPT is at most $\Gamma(n^A)^{1-1/\alpha}s^O$, which occurs when all of the speed is allocated to the job with the largest weight in (2.9).

Let $l \geq 0$ be the number of zero terms in the sum (2.9), corresponding to jobs on which PS is leading OPT. The sum in (2.9) contains $n^A - l$ non-zero terms, each decreasing due to PS at rate $i^{1-1/\alpha}dq^A/dt = i^{1-1/\alpha}s^A/n^A$. The sum is minimized (in magnitude) if these are terms $i = 1, \ldots, n^A - l$. Thus, the change in $\Phi$ due to PS is at least as negative as

$$\begin{aligned} -\Gamma \sum_{i=1}^{n^A-l} i^{1-1/\alpha}\frac{s^A}{n^A} &\leq -\Gamma \int_0^{n^A-l} i^{1-1/\alpha}\frac{s^A}{n^A}\,di \\ &\leq -\Gamma\frac{\alpha}{2\alpha-1}(n^A-l)^{2-1/\alpha}\beta^{1/\alpha}(n^A)^{(1/\alpha)-1} \end{aligned} \tag{2.28}$$

since $s^A \geq (n^A\beta)^{1/\alpha}$. This gives

$$\frac{d\Phi}{dt} \leq \Gamma(n^A)^{1-1/\alpha}s^O - \Gamma\frac{\alpha\beta^{1/\alpha}}{2\alpha-1}(n^A)^{(1/\alpha)-1}(n^A-l)^{2-1/\alpha}$$

Moreover, since $(s^A)^\alpha/\beta \leq \eta n^A$ and $l \leq n^O$, we have $n^A + (s^A)^\alpha/\beta \leq (1+\eta)n^A$ and $n^O + (s^O)^\alpha/\beta \geq l + (s^O)^\alpha/\beta$. To show (2.10), it is sufficient to show that

$$(1+\eta)n^A + \Gamma(n^A)^{1-1/\alpha}s^O - \Gamma\frac{\alpha\beta^{1/\alpha}(n^A)^{(1/\alpha)-1}(n^A-l)^{2-1/\alpha}}{2\alpha-1} \leq c(l + (s^O)^\alpha/\beta).$$

Since $n^A > 0$, dividing by $n^A$ gives the sufficient condition

$$0 \leq c(s^O)^\alpha/(\beta n^A) - \Gamma s^O/(n^A)^{1/\alpha} + cl/n^A + \Gamma\frac{\alpha\beta^{1/\alpha}}{2\alpha-1}(1-l/n^A)^{2-1/\alpha} - (1+\eta). \tag{2.29}$$

To find a sufficient condition on $c$, we take the minimum of the right hand side with respect to $s^O$, $l$ and $n^A$. Following [35], note that the minimum of the first two terms with respect to $s^O$ occurs for $s^O = (\frac{\beta\Gamma}{c\alpha})^{1/(\alpha-1)}(n^A)^{1/\alpha}$, at which point the first two terms become

$$-\left(1-\frac{1}{\alpha}\right)\left(\frac{\beta\Gamma^\alpha}{c\alpha}\right)^{1/(\alpha-1)}. \tag{2.30}$$

Now consider a lower bound on the sum of the terms in $l$. Let $j = l/n^A$, and minimize this with respect to $j \geq 0$. Setting the derivative with respect to $j$ to 0 gives $c = \beta^{1/\alpha}\Gamma(1-j)^{1-1/\alpha}$. Hence the minimum for $j \geq 0$ is for $j = 1 - (\min(1, c/(\beta^{1/\alpha}\Gamma)))^{\alpha/(\alpha-1)}$. For $c \geq \beta^{1/\alpha}\Gamma$, the sum of the terms in $l$ achieves a minimum (with respect to $l$) of $\beta^{1/\alpha}\Gamma\alpha/(2\alpha-1)$ at $l = 0$, for all $n^A$. In this case, it is sufficient that

$$0 \leq -\left(1-\frac{1}{\alpha}\right)\left(\frac{\beta\Gamma^\alpha}{c\alpha}\right)^{1/(\alpha-1)} + \beta^{1/\alpha}\Gamma\frac{\alpha}{2\alpha-1} - (1+\eta).$$

Rearranging shows that it is sufficient that $c \geq \beta^{1/\alpha}\Gamma$ and

$$\begin{aligned} c &\geq \beta\left(\frac{\Gamma}{\alpha}\right)^\alpha\left(\frac{(\alpha-1)(2\alpha-1)}{\alpha\beta^{1/\alpha}\Gamma-(1+\eta)(2\alpha-1)}\right)^{\alpha-1} \\ &= (1+\eta)\left(\frac{2\alpha-1}{\alpha}\right)^\alpha. \end{aligned}$$

where the equality uses $\Gamma = (1+\eta)(2\alpha-1)/\beta^{1/\alpha}$. □

## Appendix 2.C Proof of unfairness of SRPT

The following lemmas establish Theorem 2.5.

We start by characterizing the limiting slowdown in terms of the average speed in a system with a permanent customer.

**Lemma 2.6.** *Consider a GI/GI/1 queue with service discipline $\pi \in \{PS, SRPT\}$ with controllable service rate $s_n^\pi$ such that $\bar{s}^{\pi+1} > \rho$, then*

$$\lim_{x\to\infty}\frac{T^\pi(x)}{x} =_{a.s.} \frac{1}{\bar{s}^{\pi+1}-\rho}.$$

*Proof.* We prove only the case of PS. The proof for SRPT is analogous.

Consider a PS+1 system. Let $S(t)$ be the total work completed (service given) by time $t$. Let $R(t)$ be the service given to the permanent job by time $t$ and $\bar{R}(t)$ be the service given to all other jobs by time $t$. Note $S(t) = R(t) + \bar{R}(t)$.

Since the system is stable, we have $\lim_{t\to\infty}\bar{R}(t)/t =_{a.s.} \rho$, and by definition, we have $\lim_{t\to\infty} S(t)/t =_{a.s.}$

$\bar{s}^{PS+1}$. Thus,

$$\lim_{t\to\infty} R(t)/t =_{a.s.} \bar{s}^{PS+1} - \rho.$$

To complete the proof note that $R(T(x)) = x$ and so

$$\lim_{x\to\infty} \frac{T(x)}{x} = \lim_{t\to\infty} \frac{t}{R(t)} =_{a.s.} \frac{1}{\bar{s}^{PS+1} - \rho}.$$

□

Next, we focus on relating the length of the busy periods in the PS and SRPT systems. This eventually lets us conclude that $\bar{s}^{PS+1} > \bar{s}^{SRPT+1}$. Let $t^{\pi}(w)$ be the time when $w$ work has been completed under policy $\pi$.

**Lemma 2.7.** *Consider a single server with a controllable service rate. Assume $s_n^{PS} \geq s_n^{SRPT}$ for all $n$ and that $s_n^{PS}$ and $s_n^{SRPT}$ are both weakly monotonically increasing. Then*

$$t^{PS}(w) \leq t^{SRPT}(w).$$

*Hence, busy periods are longer under SRPT than under PS.*

*Proof.* We prove the result only in the case where $s_n^{PS} = s_n^{SRPT}$ for all $n$ and $s_n^{PS}$ and $s_n^{SRPT}$ are both strictly monotonically increasing; the general proof is analogous.

Observe that $t(w)$ is continuous under both PS and SRPT. So, it is sufficient to show that at every point $v$ when $t^{PS}(v) = t^{SRPT}(v)$ ceases to be true it is because $t^{PS}(v^+) < t^{SRPT}(v^+)$. Thus, we induct over moments when $t^{PS}(v) = t^{SRPT}(v)$ and $t^{PS}(w) \leq t^{SRPT}(w)$ for all $w \leq v$.

The base case follows from noting that $t^{PS}(0) = t^{SRPT}(0) = 0$ and that $s_n^{PS} = s_n^{SRPT}$ until the moment of the first completion, which happens under SRPT due to the optimality of SRPT. Let $w_0$ be the work that has been completed at this moment. Then, if the system is not empty, $t^{PS}(w_0^+) < t^{SRPT}(w_0^+)$ since $s_n$ is strictly monotonically increasing.

Next, consider a point $v$ such that $t^{PS}(v) = t^{SRPT}(v)$ and $t^{PS}(w) \leq t^{SRPT}(w)$ for all $w \leq v$. There are three cases:

**$n^{PS}(t^{PS}(v)) > n^{SRPT}(t^{SRPT}(v))$:** In this case, $t^{PS}(v^+) < t^{SRPT}(v^+)$ since $s_n$ is strictly increasing.

**$n^{PS}(t^{PS}(v)) = n^{SRPT}(t^{SRPT}(v))$:** In this case, $t^{PS}(w) = t^{SRPT}(w)$ for all $w > v$ until the next completion moment, $w_0$. Applying Lemma 2.8 below, we know that this completion happens under SRPT. So, $t^{PS}(w_0^+) < t^{SRPT}(w_0^+)$ since $s_n$ is strictly increasing.

**$n^{PS}(t^{PS}(v)) < n^{SRPT}(t^{SRPT}(v))$:** Lemma 2.8, below, proves that this cannot happen.

□

**Lemma 2.8.** *Consider a single server with a controllable service rate. At moments when* $t^{PS}(v) = t^{SRPT}(v)$ *and* $t^{PS}(w) \le t^{SRPT}(w)$ *for all* $w \le v$,

$$n^{PS}(t^{PS}(v)) \ge n^{SRPT}(t^{SRPT}(v)).$$

*Proof.* The first step is to warp time separately for each system such that the server is always working at rate 1 until $v$ work has been done. To do that, scale time by $1/s_{n(t)}$ at all times $t$ until that point. The warping, and hence the arrival instance, will be different in each system. Call the resulting instances $I^{PS}$ and $I^{SRPT}$.

These two instances satisfy the following relationships:

(i) The number of arrivals is the same in both instances.

(ii) The size of the $i$th arrival is the same in both instances for all $i$.

(iii) The inter-arrival times of the two instances may differ.

(iv) The $i$th arrival in $I^{SRPT}$ happens no later than the $i$th arrival in $I^{PS}$. This follows from the hypothesis of the lemma that $t^{PS}(w) \le t^{SRPT}(w)$ for all $w \le v$.

Let $n^{\pi}(t, I)$ denote the number in system at time $t$ in instance $I$ under policy $\pi$. Now, it is enough to prove that $n^{PS}(t, I^{PS}) \ge n^{SRPT}(t, I^{SRPT})$. Intuitively, this should be true because (i) the arrivals are happening earlier in $I^{SRPT}$ and (ii) SRPT minimizes the queue length.

To prove this formally, note that the optimality of SRPT immediately gives $n^{PS}(t, I^{PS}) \ge n^{SRPT}(t, I^{PS})$. Second, consider $C^{\pi}(t, I)$, the number of completions by time $t$ under policy $\pi$ and instance $I$. To finish the proof, we claim that $C^{SRPT}(t, I^{PS}) \le C^{SRPT}(t, I^{SRPT})$, whence $n^{SRPT}(t, I^{PS}) \ge n^{SRPT}(t, I^{SRPT})$. To prove the claim, first define a (non–work-conserving) policy $Q$, which, when run on instance $I^{SRPT}$, has exactly the same completion instants as SRPT does on instance $I^{PS}$. Such a $Q$ exists since all arrivals happen no later under $I^{SRPT}$ than $I^{PS}$.

By the optimality of SRPT and the definition of $Q$,

$$C^{SRPT}(t, I^{PS}) = C^{Q}(t, I^{SRPT}) \le C^{SRPT}(t, I^{SRPT}),$$

which completes the proof of the claim and the lemma. □

Finally, we use the above results to prove that $\bar{s}^{SRPT+1} < \bar{s}^{PS+1}$, which completes the proof of Theorem 2.5.

**Lemma 2.9.** *Consider a GI/GI/1 queue with a controllable service rate and an unbounded inter-arrival time distribution. Assume* $s_n^{PS} \ge s_n^{SRPT}$ *for all* $n$ *and* $s_n^{PS}$ *and* $s_n^{SRPT}$ *are both weakly*

*monotonically increasing with* $s_1 > 0$. *Further, assume that* $\sup_n s_n^{PS} > \rho$ *and* $\sup_n s_n^{SRPT} > \rho$. *Then both systems are stable and*

$$\rho < \bar{s}^{SRPT+1} < \bar{s}^{PS+1}.$$

*Proof.* First, note that the stability of $\pi$ and $s_1^\pi > 0$ guarantees that $\bar{s}_n^{\pi+1} > \rho$ for $\pi \in \{SRPT, PS\}$.

We will prove the result in the case that $s_n^{PS} = s_n^{SRPT}$ for all $n$. The case of $s_n^{PS} \geq s_n^{SRPT}$ follows immediately. We refer to the PS+1 or the SRPT+1 system as "empty" when it is empty except for the permanent customer. Define a renewal point as occurring when both the PS+1 and SRPT+1 systems are "empty". Since both systems are stable and the inter-arrival time distribution is unbounded, there are infinitely many such renewals.

At the moments when both systems are "empty", the same customers have been completed in both systems. The only difference is how much work has been done on the permanent customer. So, the policy which has completed the most of the permanent customer has the largest average service rate.

We now focus on a single renewal and determine the time-average speeds under SRPT+1 and PS+1. By the renewal reward theorem, this is enough to prove the lemma. Let time 0 be when the sub-busy period began (when both systems were last empty). Let $t_e$ be the first time when both systems are "empty". Let $v$ be the amount of work completed on the permanent customer under PS during this sub-busy period.

Now, consider an instance that is unchanged except that the permanent customer is replaced by a job of size $z$ that arrives at time 0 and that no new arrivals occurs after time $t_e$. Choose $z$ large enough that it will always have lowest priority in the SRPT+1 system. By Lemma 2.7, this busy period is at least as long under SRPT+1 as under PS+1. Thus, since $s_1^{PS} = s_1^{SRPT}$, at time $t_e$ SRPT+1 must have completed no more than $v$ work on the permanent customer. So, within this renewal, $\bar{s}^{SRPT+1} \leq \bar{s}^{PS+1}$.

Finally, the stronger statement $\bar{s}^{SRPT+1} < \bar{s}^{PS+1}$ holds since with positive probability the renewal will include exactly 2 arrivals that are both in the system at the moment of the first departure under SRPT+1, which guarantees that this completion instant is strictly earlier under SRPT+1 than under PS+1. □

# Chapter 3

# Dynamic Capacity Provisioning in Data Centers

Energy costs represent a significant fraction of a data center's budget [57] and this fraction is expected to grow as the price of energy increases in coming years. Hence, there is a growing push to improve the energy efficiency of the data centers behind cloud computing. A guiding focus for research into "green" data centers is the goal of designing data centers that are "power-proportional", i.e., use power only in proportion to the load. However, current data centers are far from this goal – even today's energy-efficient data centers consume almost half of their peak power when nearly idle [16].

A promising approach for making data centers more power-proportional is using software to dynamically adapt the number of active servers to match the current workload, i.e., to dynamically 'right-size" the data center. Specifically, dynamic right-sizing refers to adapting the way requests are dispatched to servers in the data center so that, during periods of low load, servers that are not needed do not have jobs routed to them and thus are allowed to enter a power-saving mode (e.g., go to sleep or shut down).

Technologies that implement dynamic right-sizing are still far from standard in data centers due to a number of challenges. First, servers must be able to seamlessly transition into and out of power-saving modes while not losing their state. There has been a growing amount of research into enabling this in recent years, dealing with virtual machine state [39], network state [37] and storage state [108, 5]. Second, such techniques must prove to be reliable, since administrators we talk to worry about wear-and-tear consequences of such technologies. Third, and the challenge that this chapter addresses, it is unclear how to determine the number of servers to toggle into power-saving mode and how to control servers and requests due to the lack of knowledge about future workloads, which means that a server that is put to sleep may soon need to be woken again. Although receding horizon control has been proposed for the online control [116, 78], as shown in Section 3.2, its performance really depends on the prediction window and the toggling cost of servers, which may vary widely in different data centers.

A goal of this chapter is to provide a new algorithm to address this challenge for general settings. To this end, we develop a general model that captures the major issues that affect the design of a right-sizing algorithm, including: the cost (lost revenue) associated with the increased delay from using fewer servers, the energy cost of maintaining an active server with a particular load, and the cost incurred from toggling a server into and out of a power-saving mode (including the delay, energy, and wear-and-tear costs).

This chapter makes three contributions: First, we analytically characterize the optimal offline solution (Section 3.3). We prove that it exhibits a simple, "lazy" structure when viewed in reverse time.

Second, we introduce and analyze a novel, practical online algorithm motivated by this structure (Section 3.4). The algorithm, named *lazy capacity provisioning* (LCP($w$)), uses a prediction window of length $w$ of future arrivals and mimics the "lazy' structure of the optimal algorithm, but proceeding forward instead of backwards in time. We prove that LCP($w$) is 3-competitive, i.e., its cost is at most 3 times that of the optimal offline solution. This is regardless of the workload and for very general energy and delay cost models, even when no information is used about arrivals beyond the current time period ($w = 0$). Further, in practice, LCP($w$) is far better than 3-competitive, incurring nearly the optimal cost.

Third, we validate our algorithm using two load traces (from Hotmail and a Microsoft Research data center) to evaluate the cost savings achieved from dynamic right-sizing in practice (Section 3.5). We contrast our new algorithm with receding horizon control and confirm that our algorithm provides much more stable cost saving. We show that significant savings are possible under a wide range of settings and that savings become dramatic when the workload is predictable over an interval proportional to the toggling cost. The magnitude of the potential savings depends primarily on the peak-to-mean ratio (PMR) of the workload, with a PMR of 3 being enough to give 30% cost saving. In the context of these real traces, we also discuss when it does and when it does not make sense to use dynamic right-sizing versus the alternative of "valley-filling", i.e., using periods of low load to run background/maintenance tasks. We find that dynamic right-sizing provides more than 15% cost savings even when the background work makes up 40% of the workload when the PMR is larger than 3.

## 3.1 Model and notation

Right-sizing problems vary between data centers: some systems have flexible quality of service (QoS) requirement, others have hard service level agreements (SLAs); the electricity price may be time-varying or fixed; workloads can be homogeneous or heterogeneous, etc. We first introduce a general model which captures the main issues in many right-sizing problems in different systems. We then

give examples and show how they fit into this general model.

### 3.1.1 General model

We now describe the general model used to explore the cost savings possible from dynamic right-sizing. An instance consists of a constant $\beta > 0$, a horizon[1] $T > 0$, a sequence of non-negative convex functions $g_t(\cdot)$ for $t = 1, \ldots, T-1$; $g_t(\cdot)$ can take on the value $\infty$ but cannot be identically $\infty$. For notational simplicity, let $g_T(x) \equiv 0$ and $x_0 = x_T = 0$. The model is:

$$\begin{aligned} \underset{x_1,\ldots,x_{T-1}}{\text{minimize}} \quad & \sum_{t=1}^{T} g_t(x_t) + \beta \sum_{t=1}^{T} (x_t - x_{t-1})^+ \\ \text{subject to} \quad & x_t \geq 0 \end{aligned} \tag{3.1}$$

where $(x)^+ = \max(0, x)$, and $\{x_t\}$ are non-negative scalar variables. The solution to optimization (3.1) may not be unique. By Corollary 3.1 in Appendix 3.A, there exists a solution that is the element-wise maximum. Unless otherwise stated, "the solution" refers to this maximum solution. We discuss the extension to vector $x_t$ in the next chapter.

To apply this to data center right-sizing, recall the major costs of right-sizing: (a) the cost associated with the increased delay from using fewer servers and the energy cost of maintaining an active server with a particular load; (b) the cost incurred from toggling a server into and out of a power-saving mode (including the delay, migration, and wear-and-tear costs). We call the first part "operating cost" and the second part "switching cost" and consider a discrete-time model where the timeslot length matches the timescale at which the data center can adjust its capacity. There is a (possibly long) time-interval of interest $t \in \{0, 1, \ldots, T\}$ and the capacity (i.e., number of active servers) at time $t$ is $x_t$.

The operating cost in each timeslot is modelled by $g_t(x_t)$, which presents the cost of using $x_t$ servers ($x_t$ has a vector value if servers are heterogeneous) to serve requests at timeslot $t$. Moreover, $g_t(\cdot)$ captures many other factors in timeslot $t$, including the arrival rate, the electricity price, the cap on available servers, the SLA constraints.

For the switching cost, let $\beta$ be the cost to transition a server from the sleep state to the active state and back again. We deem this cost to be incurred only when the server wakes up. Thus the switching cost for changing the number of active servers from $x_{t-1}$ to $x_t$ is $\beta(x_t - x_{t-1})^+$. The constant $\beta$ includes the costs of (i) the energy used, (ii) the delay in migrating connections/data/etc. (e.g., by VM techniques), (iii) increased wear-and-tear on the servers, and (iv) the risk associated with server toggling. If only (i) and (ii) matter, then $\beta$ is either on the order of the cost to run a server for a few seconds (waking from suspend-to-RAM or migrating network state [37] or storage

[1] If parameters such as $\beta$ are known in advance, then the results in this chapter can be extended to a model in which each instance has a finite duration, but the cost is summed over an infinite horizon.

state [108]), or several minutes (to migrate a large VM [39]). However, if (iii) is included, then $\beta$ becomes on the order of the cost to run a server for an hour [27]. Finally, if (iv) is considered then our conversations with operators suggest that their perceived risk that servers will not turn on properly when toggled is high, so $\beta$ may be many hours' server costs. Throughout, denote the operating cost of a vector $X = (x_1, \ldots, x_T)$ by $cost_o(X) = \sum_{t=1}^{T} g_t(x_t)$, the switching cost by $cost_s(X) = \beta \sum_{t=1}^{T} (x_t - x_{t-1})^+$, and $cost(X) = cost_o(X) + cost_s(X)$.

Constraints on $x_t$ are handled implicitly in functions $g_t(\cdot)$ by *extended-value extension*, i.e., defining $g_t(\cdot)$ to be $\infty$ outside its domain. This extension makes our description/notation more clear without introducing particular constraints in different data centers.

Formulation (3.1) makes a simplification that it does not enforce that $x_t$ be integer valued. This is acceptable since the number of servers in a typical data center is large. This model also ignores the requirement to maintain availability guarantees. Such issues are beyond the scope of this thesis, however previous work shows that solutions are possible [108].

This optimization problem would be easy to solve *offline*, i.e., given functions $g_t(\cdot)$ for all $t$. However, our goal is to find *online* algorithms for this optimization, i.e., algorithms that determine $x_\tau$ using only information up to time $\tau + w$, where the "prediction window" $w \geq 0$ is part of the problem instance. Here, we assume that the predictions $[g_\tau, \ldots, g_{\tau+w}]$ are known perfectly at time $\tau$, but we show in Section 3.5 that our algorithm is robust to this assumption in practice.

We evaluate the performance of an online algorithm $\mathcal{A}$ using the standard notion of *competitive ratio*. The competitive ratio of $\mathcal{A}$ is defined as the supremum, taken over all possible inputs, of $cost(\mathcal{A})/cost(OPT)$, where $cost(\mathcal{A})$ is the objective function of (3.1) under $\mathcal{A}$ and $OPT$ is the optimal offline algorithm. The analytic results of Sections 3.3 and 3.4 assume that the service has a finite duration, i.e., $T < \infty$, but hold for arbitrary sequences of convex functions $g_t(\cdot)$. Thus, the analytic results provide worst-case guarantees. However, to provide realistic cost estimates, we consider case studies in Section 3.5 where $g_t(\cdot)$ is based on real-world traces.

### 3.1.2 Special cases

Now let us consider two choices of $g_t(\cdot)$ to fit two different right-sizing problems into Formulation (3.1). The discrete-time model consists of finitely many times $t \in \{0, 1, \ldots, T\}$, where $T$ may be a year, measured in timeslots of 10 minutes. Assume the workload affects the operating cost at timeslot $t$ only by its mean arrival rate, denoted $\lambda_t$. (This may have a vector value if there are multiple types of work). This assumption is reasonable for many services such as web search, social network or email service since the request interarrival times and the response times are much shorter than the timeslots so that the provisioning can be based on the arrival rate.

We model a data center as a collection of homogeneous servers (The heterogeneous systems will be studied in the next chapter.) and focus on determining $x_t$, the number of active servers during

each time slot $t$. For notational simplicity, we assume a load balancer assigns arriving jobs to active servers uniformly, at rate $\lambda_t/x_t$ per server (which is widely deployed and turns out to be optimal for many systems). Assume the power of an active server handling arrival rate $\lambda$ is $e(\lambda)$, and the performance metric we care about is $d(\lambda)$ (e.g., average response time or response time violation probability). For example, a common model of the power for typical servers is an affine function $e(\lambda) = e_0 + e_1\lambda$ where $e_0$ and $e_1$ are constants; e.g., see [1]. The average response time can be modeled using standard queuing theory results. If the server happens to be modeled by an M/GI/1 processor sharing queue then the average response time is $d(\lambda) = 1/(\mu-\lambda)$, where the service rate of the server is $\mu$ [75]. Other examples include, for instance, using the 99th percentile of delay instead of the mean. In fact, if the server happens to be modeled by an M/M/1 processor sharing queue then the 99th percentile is $\log(100)/(\mu-\lambda)$, and so the form of $d(\lambda)$ does not change [75]. Note that, in practice, $e(\lambda)$ and $d(\lambda)$ can be empirically measured by observing the system over time without assuming mathematical models for them. Our analytical results work for any $e(\cdot)$ and $d(\cdot)$ as long as they result in convex $g_t(\cdot)$. But for simplicity, we will use $e(\lambda) = e_0 + e_1\lambda$ and $d(\lambda) = 1/(\mu - \lambda)$ for our numerical experiments.

**Example 1: Services with flexible QoS requirement**

Many Internet services, such as web search, email service and social network are flexible in response time in certain range. For such applications, it is hard to find a threshold to distinguish "good services" and "bad services". Instead, the revenue is lost gradually as the response time increases and thus our goal is to balance the performance and the power cost [110, 43, 78, 36]. One natural model for the lost revenue is $d_1\lambda(d(\lambda) - d_0)^+$ where $d_0$ is the minimum delay users can detect and $d_1$ is a constant. This measures the perceived delay weighted by the fraction of users experiencing that delay. For the energy cost, assume the electricity price is $p_t$ at timeslot $t$, then the energy cost is $p_t e(\lambda)$. There may be a time-varying cap $M_t$ on the number of active server available due to other activities/services in the data center (e.g., backup activities). The combination together with the dispatching rule at the load balancer gives

$$\begin{aligned} \text{minimize } & \sum_{t=1}^{T} x_t \left( \frac{d_1\lambda_t}{x_t} \left( d\left(\frac{\lambda_t}{x_t}\right) - d_0 \right)^+ + p_t e\left(\frac{\lambda_t}{x_t}\right) \right) + \beta \sum_{t=1}^{T} (x_t - x_{t-1})^+ \qquad (3.2) \\ \text{subject to } & \lambda_t \le x_t \le M_t \end{aligned}$$

Compared to Formulation (3.1), we have

$$g_t(x_t) = \begin{cases} x_t \left( \frac{d_1\lambda_t}{x_t} \left( d\left(\frac{\lambda_t}{x_t}\right) - d_0 \right)^+ + p_t e\left(\frac{\lambda_t}{x_t}\right) \right) & \text{if } \lambda_t \le x_t \le M_t, \\ \infty & \text{otherwise.} \end{cases}$$

We can see that $g_t(x_t)$ is in the form of the perspective function of $f(z) = d_1 z(d(z)-d_0)^+ + p_t e(z)$, which is convex under common models of $d(\cdot)$ and $e(\cdot)$, such as M/G/1 queueing models with or without speed scaling of individual servers. Thus $g_t(\cdot)$ is convex under common models.

**Example 2: Services with hard QoS constraints**

Some data centers have to support services with hard constraints such as SLA requirement or delay constraints for multimedia applications. For such systems, the goal is to minimize energy cost while satisfy the constraints, as shown in [104, 115]. The optimization becomes

$$\begin{aligned} &\text{minimize } \sum_{t=1}^{T} x_t p_t e\left(\frac{\lambda_t}{x_t}\right) + \beta \sum_{t=1}^{T} (x_t - x_{t-1})^+ \\ &\text{subject to } x_t \geq \lambda_t \text{ and } d\left(\frac{\lambda_t}{x_t}\right) \leq D_t \end{aligned} \tag{3.3}$$

where $d(\cdot)$ can be average delay, delay violation probability or other performance metrics. This constraint usually defines a convex domain for $x_t$. Actually if we assume that $d(\cdot)$ is an increasing function of the load, then it is equivalent to imposing an upper bound on the load per server (i.e., $\lambda_t/x_t$ is less than certain threshold) and thus results in a convex optimization problem.

The corresponding $g_t(\cdot)$ can be defined as follows:

$$g_t(x_t) = \begin{cases} x_t p_t e\left(\frac{\lambda_t}{x_t}\right) & \text{if } x_t \geq \lambda_t \text{ and } d\left(\frac{\lambda_t}{x_t}\right) \leq D_t, \\ \infty & \text{otherwise.} \end{cases}$$

## 3.2 Receding horizon control

As motivation for deriving a new algorithm, let us first consider the standard approach. An online policy that is commonly proposed to control data centers [116, 78] (and other dynamic systems) is receding horizon control (RHC) algorithms, also known as model predictive control. RHC has a long history in the control theory literature [79, 80, 90] where the focus was on stability analysis. In this section, let us introduce briefly the competitive analysis result of RHC for our data center problem (The result will be proven in the next chapter where we study the performance of RHC in both homogeneous systems and heterogeneous systems.) and see why we may need a better online algorithm.

Informally, RHC($w$) works by solving, at time $\tau$, the cost optimization over the window $(\tau, \tau+w)$ given the starting state $x_{\tau-1}$, and then using the first step of the solution, discarding the rest.

Formally, define $X^\tau(x_{\tau-1}; g_\tau \dots g_{\tau+w})$ as the vector in $\mathbb{R}^{w+1}$ indexed by $t \in \{\tau, \dots, \tau+w\}$, which

is the solution to

$$\begin{aligned} \text{minimize} \quad & \sum_{t=\tau}^{\tau+w} g_t(x_t) + \beta \sum_{t=\tau}^{\tau+w} (x_t - x_{t-1})^+ \\ \text{subject to} \quad & x_t \geq 0 \end{aligned} \tag{3.4}$$

Then, RHC($w$) works as follows.

**Algorithm** (Receding horizon control: RHC($w$))**.** *For all $t \leq 0$, set the number of active servers to $x_t = 0$. At each timeslot $\tau \geq 1$, set the number of active servers to*

$$x_\tau = X_\tau^\tau(x_{\tau-1}; g_\tau \dots g_{\tau+w})$$

Note that (3.4) need not have a unique solution. We define RHC($w$) to select the solution with the greatest first entry. Define $e_0$ the minimum cost per timeslot for an active server, then we have the following theorem:

**Theorem.** *RHC($w$) is $(1 + \frac{\beta}{(w+1)e_0})$-competitive for optimization (3.1) but not better than $(\frac{1}{w+2} + \frac{\beta}{(w+2)e_0})$-competitive.*

This highlights that, with enough lookahead, RHC($w$) is guaranteed to perform quite well. However, its competitive ratio depends on the parameters $\beta$, $e_0$ and $w$. These parameters vary widely in different data centers since they host different services or have different SLAs. Thus, RHC($w$) may have a poor competitive ratio for a data center with big switching cost or small prediction window.

This poor performance is to be expected since RHC($w$) makes no use of the structure of the optimization we are trying to solve. We may wonder if there exists better online algorithm for this problem that performs well for a wider range of parameters, even when the prediction window $w = 0$ (no workload prediction except current timeslot).

## 3.3 The optimal offline solution

In order to exploit the structure of the specific problem (3.1), the first natural task is to characterize the optimal offline solution, i.e., the optimal solution given $g_t(\cdot)$ for all $t$. The insight provided by the characterization of the offline optimum motivates the formulation of our online algorithm.

It turns out that there is a simple characterization of the optimal offline solution $X^*$ to the optimization problem (3.1), in terms of two bounds on the optimal solution which correspond to charging cost $\beta$ either when a server comes out of power-saving mode (as (3.1) states) or when it goes in. The optimal $x_\tau^*$ can be viewed as "lazily" staying within these bounds going backwards in time.

More formally, let us first describe lower and upper bounds on $x_\tau^*$, denoted $x_\tau^L$ and $x_\tau^U$, respectively. Let $(x_{\tau,1}^L, \ldots, x_{\tau,\tau}^L)$ be the solution vector to the optimization problem

$$\begin{aligned} \text{minimize} \quad & \sum_{t=1}^{\tau} g_t(x_t) + \beta \sum_{t=1}^{\tau} (x_t - x_{t-1})^+ \\ \text{subject to} \quad & x_t \geq 0 \end{aligned} \tag{3.5}$$

where $x_0 = 0$. Then, define $x_\tau^L = x_{\tau,\tau}^L$. Similarly, let $(x_{\tau,1}^U, \ldots, x_{\tau,\tau}^U)$ be the solution vector to the optimization

$$\begin{aligned} \text{minimize} \quad & \sum_{t=1}^{\tau} g_t(x_t) + \beta \sum_{t=1}^{\tau} (x_{t-1} - x_t)^+ \\ \text{subject to} \quad & x_t \geq 0 \end{aligned} \tag{3.6}$$

Then, define $x_\tau^U = x_{\tau,\tau}^U$.

Notice that in each case, the optimization problem includes only times $1 \leq t \leq \tau$, and so ignores the future information for $t > \tau$. In the case of the lower bound, $\beta$ cost is incurred for each server toggled on, while in the upper bound, $\beta$ cost is incurred for each server toggled into power-saving mode.

**Lemma 3.1.** *For all $\tau$, $x_\tau^L \leq x_\tau^* \leq x_\tau^U$.*

Given Lemma 3.1, we now characterize the optimal solution $x_\tau^*$. Define $(x)_a^b = \max(\min(x, b), a)$ as the projection of $x$ into $[a, b]$. Then, we have:

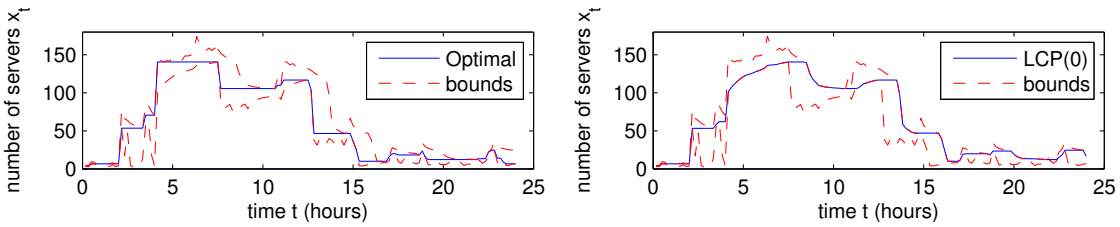
**Theorem 3.1.** *The optimal solution $X^* = (x_1^*, \ldots, x_T^*)$ of the data center optimization problem (3.1) satisfies the backward recurrence relation*

$$x_\tau^* = \begin{cases} 0, & \tau > T; \\ \left(x_{\tau+1}^*\right)_{x_\tau^L}^{x_\tau^U}, & \tau \leq T. \end{cases}$$

Theorem 3.1 and Lemma 3.1 are proven in Appendix 3.A.

An example of the optimal $x_t^*$ can be seen in Figure 3.1(a). Many more numeric examples of the performance of the optimal offline algorithm are provided in Section 3.5.

Theorem 3.1 and Figure 3.1(a) highlight that the optimal algorithm can be interpreted as moving backwards in time, starting with $x_T^* = 0$ and keeping $x_\tau^* = x_{\tau+1}^*$ unless the bounds prohibit this, in which case it makes the smallest possible change. This interpretation highlights that it is impossible for an online algorithm to compute $x_\tau^*$ since, without knowledge of the future, an online algorithm cannot know whether to keep $x_\tau$ constant or to follow the upper/lower bound.




(a) Offline optimal (b) LCP(0)

Figure 3.1: Illustrations of (a) the offline optimal solution and (b) LCP(0) for the first day of the MSR workload described in Section 3.5 with a sampling period of 10 minutes. The operating cost is defined in (3.2) with $d_0 = 1.5$, $d_1 = 1$, $\mu = 1$, $p_t = 1$, $e_0 = 1$ and $e_1 = 0$ and the switching cost has $\beta = 6$ (corresponding to the energy consumption for one hour).

## 3.4 Lazy capacity provisioning

A major contribution of this chapter is the presentation and analysis of a novel *online* algorithm, lazy capacity provisioning (LCP($w$)). At time $\tau$, LCP($w$) knows only $g_t(\cdot)$ for $t \leq \tau + w$, for some prediction window $w$. As mentioned before, we assume that these are known perfectly, but we show in Section 3.5 that the algorithm is robust to this assumption in practice. The design of LCP($w$) is motivated by the structure of the optimal offline solution described in Section 3.3. Like the optimal solution, it "lazily" stays within upper and lower bounds. However, it does this moving forward in time instead of backwards in time.

Before defining LCP($w$) formally, recall that the bounds $x_\tau^U$ and $x_\tau^L$ do not use knowledge about the loads in the prediction window of LCP($w$). To use it, define refined bounds $x_\tau^{U,w}$ and $x_\tau^{L,w}$ such that $x_\tau^{U,w} = x_{\tau+w,\tau}^U$ in the solution of (3.6) and $x_\tau^{L,w} = x_{\tau+w,\tau}^L$ in that of (3.5). Note that $w = 0$ is allowed (no future prediction) and $x_\tau^{U,0} = x_\tau^U$ and $x_\tau^{L,0} = x_\tau^L$. The following generalization of Lemma 3.1 is proven in Appendix 3.B.

**Lemma 3.2.** *$x_\tau^L \leq x_\tau^{L,w} \leq x_\tau^* \leq x_\tau^{U,w} \leq x_\tau^U$ for all $w \geq 0$.*

Now, we are ready to define LCP($w$) using $x_\tau^{U,w}$ and $x_\tau^{L,w}$.

**Algorithm 3.1** (Lazy capacity provisioning: LCP($w$))**.**
*Let $X^{LCP(w)} = (x_0^{LCP(w)}, \ldots, x_T^{LCP(w)})$ denote the vector of active servers under LCP($w$). This vector can be calculated using the following forward recurrence relation*

$$x_\tau^{LCP(w)} = \begin{cases} 0, & \tau \leq 0; \\ \left(x_{\tau-1}^{LCP(w)}\right)_{x_\tau^{L,w}}^{x_\tau^{U,w}}, & \tau \geq 1. \end{cases}$$

Figure 3.1(b) illustrates the behavior of LCP(0). Note its similarity with Figure 3.1(a), but with the laziness in forward time instead of reverse time.

The computational demands of LCP($w$) may initially seem prohibitive as $\tau$ grows, since calculating $x_\tau^{U,w}$ and $x_\tau^{L,w}$ requires solving convex optimizations of size $\tau + w$. However, it is possible to calculate $x_\tau^{U,w}$ and $x_\tau^{L,w}$ without using the full history. Lemma 3.10 in Appendix 3.B implies that it is enough to use only the history since the most recent point when the solutions of (3.5) and (3.6) are either both increasing or both decreasing, if such a point exists. In practice, this period is typically less than a day due to diurnal traffic patterns, and so the convex optimization, and hence LCP($w$), remains tractable even as $\tau$ grows. In Figure 3.1(b), each point is solved in under half a second.

Next, consider the cost incurred by LCP($w$). Section 3.5 discusses the cost in realistic settings, while in this section we focus on worst-case bounds, i.e., characterize the competitive ratio. The following theorem is proven in Appendix 3.B.

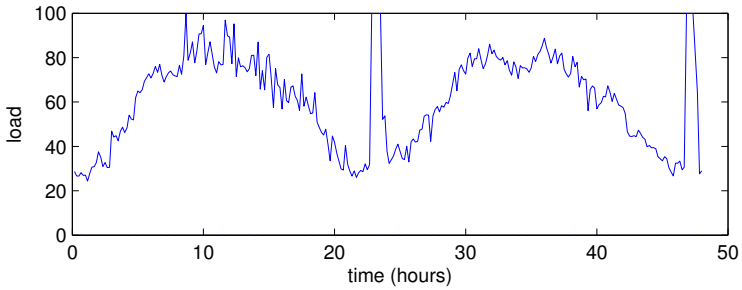
**Theorem 3.2.** *$cost(X^{LCP(w)}) \leq cost(X^*) + 2cost_s(X^*)$. Thus, LCP($w$) is 3-competitive for optimization (3.1). Further, for any finite $w$ and $\epsilon > 0$ there exists an instance, with $g_t$ of the form $x_t f(\lambda_t/x_t)$, such that LCP($w$) attains a cost greater than $3 - \epsilon$ times the optimal cost.*

Note that Theorem 3.2 says that the competitive ratio is independent of the prediction window size $w$, the switching cost $\beta$, and is uniformly bounded above regardless of the form of the operating cost functions $g_t(\cdot)$. Surprisingly, this means that even the "myopic" LCP(0) is 3-competitive, regardless of $\{g_t(\cdot)\}$, despite having no information beyond the current timeslot. It is also surprising that the competitive ratio is tight regardless of $w$. Seemingly, for large $w$, LCP($w$) should provide reduced costs. Indeed, for any particular workload, as $w$ grows the cost decreases and eventually matches the optimal, when $w > T$. However, for any fixed $w$, there is a worst-case sequence of cost functions such that the competitive ratio is arbitrarily close to 3. Moreover, there is such a sequence of cost functions having the natural form $x_t f(\lambda_t/x_t)$, which corresponds to a time-varying load $\lambda$ being shared equally among $x_t$ servers, which each has a time-invariant (though pathological) operating cost $f(\lambda)$ when serving load $\lambda$.

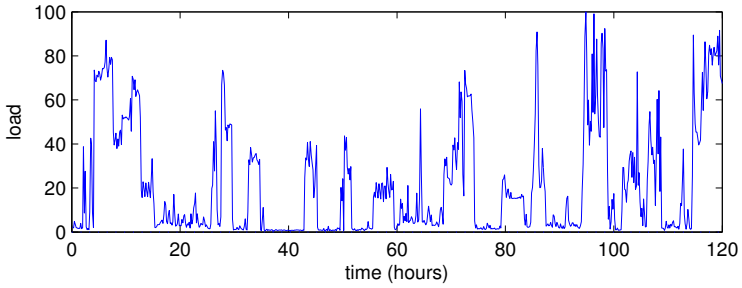
Finally, though 3-competitive may seem like a large gap, the fact that $cost(X^{LCP(w)}) \leq cost(X^*) + 2cost_s(X^*)$ highlights that the gap will tend to be much smaller in practice, where the switching costs make up a small fraction of the total costs since dynamic right-sizing would tend to toggle servers once a day due to the diurnal traffic.

## 3.5 Case studies

In this section the goal is two-fold: The first is to evaluate the cost incurred by LCP($w$) relative to the optimal solution and RHC($w$) for realistic workloads. The second is more generally to illustrate the cost savings and energy savings that come from dynamic right-sizing in data centers. To accomplish these goals, we experiment using two real-world traces.




(a) Hotmail




(b) MSR

Figure 3.2: Illustration of the traces used for numerical experiments.

### 3.5.1 Experimental setup

Throughout the experimental setup, the aim is to choose parameters that provide conservative estimates of the cost savings from LCP($w$) and right-sizing in general.

#### Cost benchmark

Current data centers typically do not use dynamic right-sizing and so to provide a benchmark against which LCP($w$) is judged, we consider the cost incurred by a "static" right-sizing scheme for capacity provisioning. This chooses a constant number of servers that minimizes the costs incurred based on full knowledge of the entire workload. This policy is clearly not possible in practice, but it provides a conservative estimate of the savings from right-sizing since it uses perfect knowledge of all peaks and eliminates the need for overprovisioning in order to handle the possibility of flash crowds or other traffic bursts.

#### Cost function parameters

The cost is of the form (3.2), characterized by the parameters $d_0$, $d_1$, $\mu$, $p_t$, $e_0$ and $e_1$, and the switching cost $\beta$. We normalize $\mu = 1$, $p_t = 1$ and choose units such that the fixed power is $e_0 = 1$. The load-dependent power is set to $e_1 = 0$, because the energy consumption of current servers is dominated by the fixed costs [16].

The delay cost $d_1$ reflects revenue lost due to customers being deterred by delay, or to violation of SLAs. We set $d_1/e_0 = 1$ for most experiments but consider a wide range of settings in Figure 3.8. The minimum perceptible delay is set to $d_0 = 1.5$ times the time to serve a single job. The value 1.5 is realistic or even conservative, since "valley filling" experiments similar to those of Section 3.5.2 show that a smaller value would result in a significant added cost when using valley filling, which operators now do with minimal incremental cost.

The normalized switching cost $\beta/e_0$ measures the duration a server must be powered down to outweigh the switching cost. We use $\beta = 6e_0$, which corresponds to the energy consumption for one hour (six samples). This was chosen as an estimate of the time a server should sleep so that the wear-and-tear of power cycling matches that of operating [27].

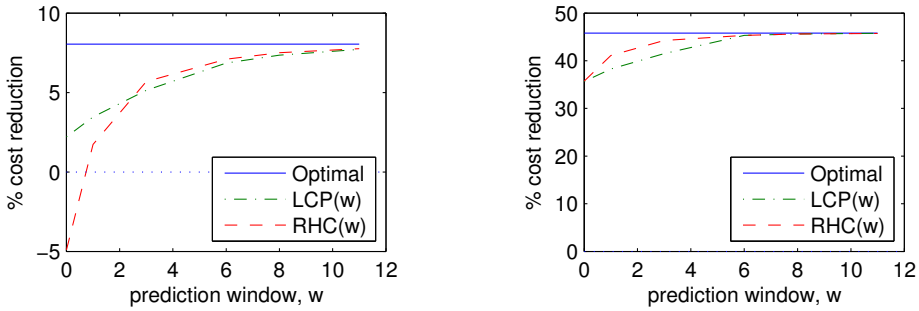
**Workload information**

The workloads for these experiments are drawn from I/O traces of two real-world data centers. The first set of traces is from Hotmail, a large email service running on tens of thousands of servers. We used traces from 8 such servers over a 48-hour period, starting at midnight (PDT) on Monday August 4 2008 [108]. The second set of traces is taken from 6 RAID volumes at MSR Cambridge. The traced period was 1 week starting from 5PM GMT on 22nd February 2007 [108]. These activity traces represent respectively a service used by millions of users and a small service used by hundreds of users. The traces are normalized such that the peak load is 100, which are shown in Figure 3.2. Notice that this normalization does not affect the experiment results since only the shape of trace matters for cost saving. Both sets of traces show strong diurnal properties and have peak-to-mean ratios (PMRs) of 1.64 and 4.64 for Hotmail and MSR respectively. Loads were averaged over disjoint 10 minute intervals.

The Hotmail trace contains significant nightly activity due to maintenance processes (backup, index creation, etc.) which is not shown fully in Figure 3.2(a). The data center, however, is provisioned for the peak foreground activity. This creates a dilemma: should our experiments include the maintenance activity or to remove it? Figure 3.5 illustrates the impact of this decision. If the spike is retained, it makes up nearly 12% of the total load and forces the static provisioning to use a much larger number of servers than if it were removed, making savings from dynamic right-sizing much more dramatic. To provide conservative estimates of the savings from right-sizing, we chose to trim the size of the spike to minimize the savings from right-sizing. This trimming makes the nightly background spike have value roughly equal to 100 (the peak foreground activity).

### 3.5.2 When is right-sizing beneficial?

Our experiments are organized in order to illustrate the impact of a wide variety of parameters on the cost savings provided by dynamic right-sizing with LCP($w$). The goal is to better understand when




(a) Hotmail (b) MSR

Figure 3.3: Impact of prediction window size on cost incurred by LCP($w$).

dynamic right-sizing can provide large enough cost savings to warrant the extra implementation complexity. Remember that throughout, we have attempted to choose experimental settings so that the benefit of dynamic right-sizing is conservatively estimated.

The analytic results show that the performance of RHC($w$) is more sensitive to the prediction window $w$ and the switching cost $\beta/e_0$ while LCP($w$) works well for general settings. The first two experiments consider the realistic cost saving from RHC($w$) and LCP($w$) with varying $w$ and $\beta/e_0$ under real-world traces.

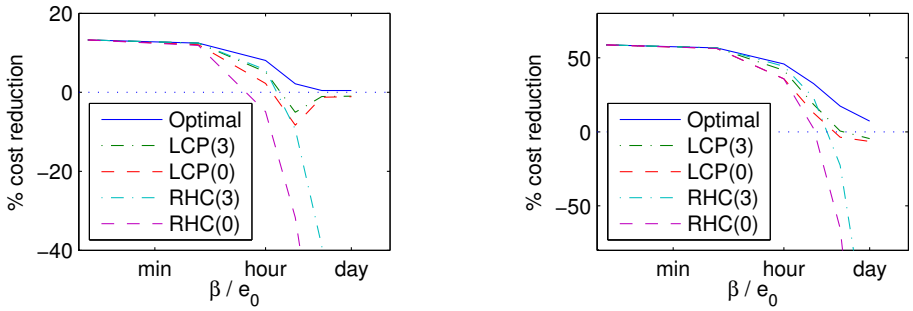
### Impact of prediction window size

The first parameter we study is the impact of the predictability of the workload. In particular, depending on the workload, the prediction window $w$ for which accurate estimates can be made could be on the order of tens of minutes or on the order of hours. Figure 3.3 illustrates its impact on the cost savings of RHC($w$) and LCP($w$), where the unit of $w$ is one timeslot of is 10 minutes.

The first observation from Figure 3.3 is that the savings possible ("Optimal") in the MSR trace are larger than in the Hotmail trace. Second, the cost saving of LCP($w$) is similar to RHC($w$) when $w \geq 3$. However, RHC($w$) can be much worse when $w \leq 2$, i.e., LCP($w$) has more stable cost saving than RHC($w$), which supports the analytic results very well. In both traces, a big fraction of the optimal cost savings is achieved by LCP(0), which uses only workload predictions about the current timeslot (10 minutes). The fact that this myopic algorithm provides significant gain over static provisioning is encouraging. Further, a prediction window that is approximately the size of $\beta = 6$ (i.e., one hour) gives nearly the optimal cost savings.

### Impact of switching costs

One of the main worries when considering right-sizing is the switching cost of toggling servers, $\beta$, which includes the delay costs, energy costs, costs of wear-and-tear, and other risks involved. Thus,




(a) Hotmail (b) MSR

Figure 3.4: Impact of switching cost, against time on a logarithmic scale.

an important question to address is: "How large must switching costs be before the cost savings from right-sizing disappears?"

Figure 3.4 shows that significant gains are possible provided $\beta$ is smaller than the duration of the valleys. Given that the energy costs, delay costs, and wear-and-tear costs are likely to be on the order of an hour, this implies that unless the risks associated with toggling a server are perceived to be extreme, the benefits from dynamic right-sizing are large in the MSR trace (high PMR case). Though the gains are smaller in the Hotmail case for large $\beta$, this is because the spike of background work splits an 8 hour valley into two short 4 hour valleys. If these tasks were shifted or balanced across the valley, the Hotmail trace would show significant cost reduction for much larger $\beta$, similarly to the MSR trace.

Another key observation is that the cost saving of LCP($w$) is similar to that of RHC($w$) when $\beta$ is small or moderate. However, when $\beta$ becomes big, LCP($w$) is much better than RHC($w$), which confirms that LCP($w$) provides more stable cost saving than RHC($w$). Since we use moderate prediction window and moderate switching cost as the default setting in our experiments, we will show only the LCP($w$) results but not RHC($w$) results for the remaining experiments.

### Prediction error

The LCP($w$) algorithm depends on having estimates for the arrival rate during the current timeslot as well as for $w$ timeslots into the future. Our analysis in Section 3.4 assumes that these estimates are perfect, but of course in practice there are prediction errors. Figure 3.6 shows the cost reduction for the Hotmail trace when additive white Gaussian noise of increasing variance is added to the predictions used by LCP($w$), including the workload in current timeslot. A variance of 0 corresponds to perfect knowledge of the near-future workload, and this figure shows that the performance does not degrade significantly when there is moderate uncertainty, which suggests that the assumptions of the analysis are not problematic. The plot for the MSR trace is qualitatively similar, although the actual cost savings are significantly larger.

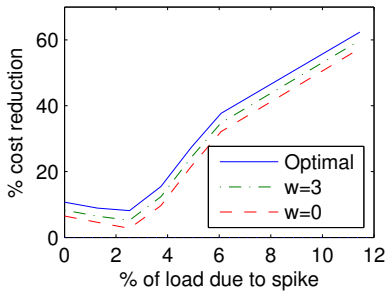



Figure 3.5: Impact of overnight peak in the Hotmail workload.

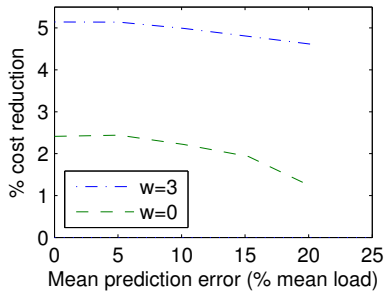


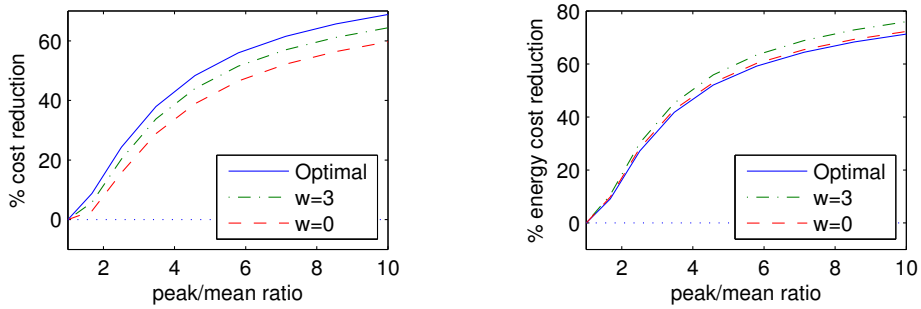

Figure 3.6: Impact of prediction error on LCP($w$) under Hotmail workload.

Even very inexact knowledge of future workloads can be beneficial. Note that LCP(0)'s upper and lower bounds $x^U$ and $x^L$ implicitly assume that future workloads will be infinite and 0, respectively. Performance can be improved if either of these bounds can be tightened. For example, it may be possible to use daily trends to predict a lower bound on the load, while the upper bound is still taken to be infinite to allow for flash crowds. Given that prediction errors for real data sets tend to be small [51, 78], based on these plots, to simplify our experiments we allow LCP($w$) perfect predictions.

**Impact of peak-to-mean ratio (PMR)**

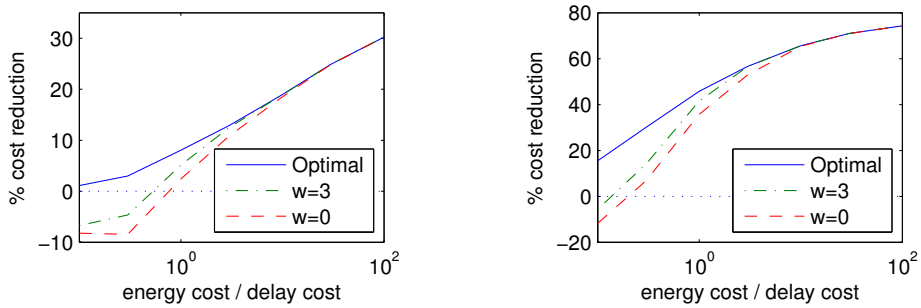
Dynamic right-sizing inherently exploits the gap between the peaks and valleys of the workload, and intuitively provides larger savings as that gap grows. To illustrate this, we artificially scaled the PMR of each trace and calculated the resulting savings. The results, in Figure 3.7, illustrate that the intuition holds for both cost savings and energy savings. The gain grows quickly from zero at PMR=1, to 5–10% at PMR$\approx$2 which is common in large data centers, to very large values for the higher PMRs common in small to medium sized data centers. This shows that, even for small data centers where the overhead of implementing right-sizing is amortized over fewer servers, there is a significant benefit in doing so. To provide some context for the monetary value of these savings, consider that a typical 50,000 server data center has an electricity bill of around \$1 million/month [57].

The workload for the figure is generated from the Hotmail workload by scaling $\lambda_t$ as $\hat{\lambda}_t = k(\lambda_t)^\alpha$, varying $\alpha$ and adjusting $k$ to keep the mean constant. Note that though Figure 3.7 includes only the results for Hotmail, the resulting plot for the MSR trace is nearly identical. This highlights that the difference in cost savings observed between the two traces is primarily due to the fact that the PMR of the MSR trace is so much larger than that of the Hotmail trace.




(a) Total cost (b) Energy cost

Figure 3.7: Impact of the peak-to-mean ratio of the workload on the total cost and energy cost incurred by LCP($w$) in the Hotmail workload.




(a) Hotmail (b) MSR

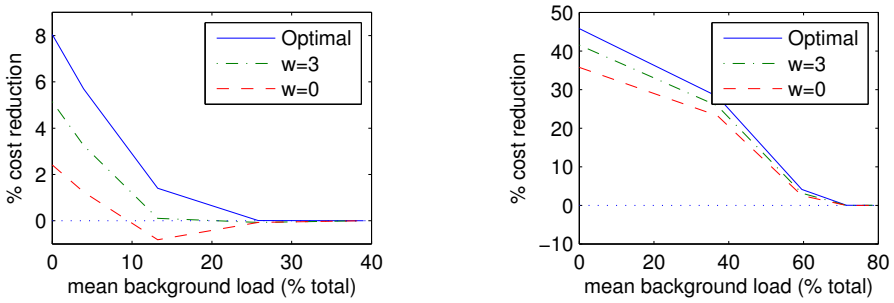
Figure 3.8: Impact of increasing energy costs.

### Impact of energy costs

Clearly the benefit of dynamic right-sizing is highly dependent on the cost of energy. As the economy is forced to move towards more expensive renewable energy sources, this cost will inevitably increase and Figure 3.8 shows how this increasing cost will affect the cost savings possible from dynamic right-sizing. Note that the cost savings from dynamic right-sizing grow quickly as energy costs rise. However, even when energy costs are quite small relative to delay costs, we see improvement in the case of the MSR workload due to its large PMR.

### Impact of valley filling

A common alternative to dynamic right-sizing that is often suggested is to run very delay-insensitive maintenance/background processes during the periods of low load, known as "valley filling". Some applications have a huge amount of such background work, e.g., search engines tuning their ranking algorithms. If there is enough such background work, then the valleys can in principle be entirely filled and so the PMR$\approx 1$ and thus dynamic right-sizing is unnecessary. Thus, an important question is: "How much background work is enough to eliminate the cost savings from dynamic right-sizing?"




(a) Hotmail (b) MSR

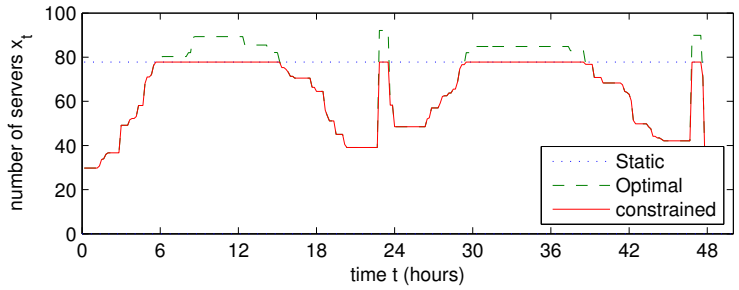
Figure 3.9: Impact of background processes. The improvement of LCP($w$) over static provisioning as a function of the percentage of the workload that is background tasks.

Figure 3.9 shows that, in fact, dynamic right-sizing provides cost savings even when background work makes up a significant fraction of the total load. For the Hotmail trace, significant savings are still possible when background load makes upwards of 10% of the total load, while for the MSR trace this threshold becomes nearly 60%. Note that Figure 3.9 results from considering "ideal" valley filling, which results in a perfectly flat load during the valleys, but does not give background processes lower queueing priority.

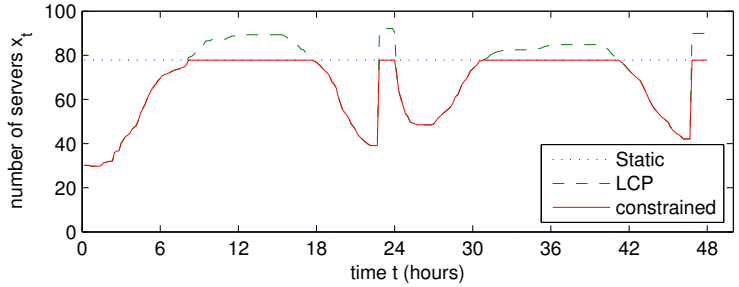
### Impact of capacity limits

Energy-related expenses account for almost half of the cost of a data center. However, many of those, such as power supply and cooling infrastructure, depend on the peak power rather than the total energy. This raises the question of whether LCP increases the peak number of servers used, or equivalently the peak power consumption. The dotted line of Figure 3.10 shows the optimal static number of servers to use, which we call $M$, while the dashed line in Figure 3.10(a) shows the optimal number and that in Figure 3.10(b) shows the number used by LCP. Dynamic provisioning uses a higher peak number of servers, which would increase the cost.

However, the LCP algorithm (and off-line optimization) can impose an additional constraint $x_t \leq M_t = M$ to ensure that the infrastructure cost is not increased. Indeed, when $M_t$ is independent of $t$ the constrained optimum appears to have the very simple form $\min(M, x_t^*)$ — the unconstrained optimum clipped to $M$ — though this structure does not hold for general time-varying bounds. An example of $x_t$ is shown by the solid lines in Figure 3.10. Despite the additional constraint, most of the potential gains are still realized by dynamic right-sizing. In this case, the gain of the optimal solution drops from 8% to 6.5%, and that of LCP drops from 5.1% to 4.4%.




(a) Offline optimal




(b) LCP(0)

Figure 3.10: Illustrations of (a) the offline optimal solution and (b) LCP(0) for the Hotmail workload described in Section 3.5 with a sampling period of 10 minutes. The operating cost is as in Figure 3.1. The dotted line shows the optimal static number of servers to use, the dashed line shows the optimal or LCP provisioning without constraints, and the solid line shows the constrained optimum or constrained LCP solution. Notice that the constrained optimum turns out to correspond to simple clipping of the unconstrained optimum.

## 3.6 Concluding remarks

This chapter has provided a new online algorithm, LCP($w$), for dynamic right-sizing in data centers. The algorithm is motivated by the structure of the optimal offline solution and guarantees cost no larger than 3 times the optimal cost, under very general settings — arbitrary workloads, and general delay cost and general energy cost models provided that they result in a convex operating cost. Further, in realistic settings the cost of LCP($w$) is nearly optimal. Additionally, LCP($w$) is simple to implement in practice and does not require significant computational overhead. Moreover, we contrast LCP($w$) with the more traditional approach of receding horizon control RHC($w$) and show that LCP($w$) provides much more stable cost saving with general settings.

Additionally, the case studies used to evaluate LCP($w$) highlight that the cost and energy savings achievable by dynamic right-sizing are significant. The case studies highlight that if a data center has PMR larger than 3, a cost of toggling a server of less than a few hours of server costs, and less than 40% background load then the cost savings from dynamic right-sizing can be conservatively estimated at larger than 15%. Thus, even if a data center is currently performing valley filling, it can still achieve significant cost savings by dynamic right-sizing.

Interest in right-sizing has been growing since [36] and [99] appeared early last decade. Approaches range from very "analytic" work focusing on developing algorithms with provable guarantees to "systems" work focusing on implementation. Early systems work such as [99] achieved substantial savings despite ignoring switching costs in their design. Other designs have focused on decentralized frameworks, e.g., [73], [88] and [68], as opposed to the centralized framework considered here. A recent survey is [18].

Related analytic work focusing on dynamic right-sizing includes [100], which reallocates resources between tasks within a data center, and [64], which considers sleep of individual components, among others. Typically, approaches have applied optimization using queueing theoretic models, e.g., [47, 46], or control theoretic approaches, e.g., [63, 38, 110]. A recent survey of analytic work focusing on energy efficiency in general is [3]. Our work is differentiated from this literature by the generality of the model considered, which subsumes most common energy and delay cost models used by analytic researchers, and the fact that we provide worst-case guarantees for the cost of the algorithm, which is typically not possible for queueing or control theoretic algorithms.

The model and algorithm introduced in this chapter most closely ties to the online algorithms literature, specifically the classic rent-or-buy (or "ski rental") problem [28]. The best deterministic strategy for deciding when to turn off a single idle server (i.e., to stop "renting" and to "buy") is 2-competitive [71]. Additionally, there is a randomized algorithm which is asymptotically $e/(e-1)$-competitive [70]. The "lower envelope" algorithm of [64], which generalizes the standard rent-or-buy algorithm, puts a device into deeper sleep mode $m$ at a time $t$ such that the optimal solution would use $m$ if the idle period finished right after $t$. This is like LCP following $x^U$ downward; in [64] a device must be fully on to serve work, and so there is no equivalent to $x^L$ forcing the system to turn on in stages.

An important difference between these simple models and this chapter is that the cost of the "rent" action may change arbitrarily over time in the data center optimization problem. Problems with this sort of dynamics typically have competitive ratios greater than 2. For example, when rental prices vary in time, the competitive ratio is unbounded in general [19]. Further, for "metrical task systems" [29], which generalize rent-or-buy problems and the data center optimization problem, there are many algorithms available, but they typically have competitive ratios that are at best poly-logarithmic in the number of servers. Perhaps the most closely related prior work from this area is the page-placement problem (deciding on which server to store a file), which has competitive ratio 3 [21]. The page replacement-problem is nearly a discrete version of the data center optimization problem where the cost function is restricted to $f(x) = |x - 1|$.

The general model proposed in this chapter captures many applications other than the right-sizing problems in data centers. Intuitively, this model seeks to minimize the sum of a sequence of convex functions while "smooth" solutions are preferred. Other applications may be captured

by this model, such as video streaming [127], in which encoding quality should vary depending on network bandwidth, but large changes in encoding quality are visually annoying to users. A natural generalization of the framework of (3.1) is to make $x_t$ a vector, corresponding to managing resources of multiple different types. This extension has many important applications, including joint capacity provision in geographically distributed data centers [88], automatically switched optical networks (ASONs) in which there is a cost for re-establishing a lightpath [126], and power generation with dynamic demand, since the cheapest types of generators typically have very high switching costs [69]. In the next chapter, we are going to study this generalization and develop online algorithms for it.

## Appendix 3.A Analysis of the offline optimal solution

In this section we will prove Lemma 3.1 and Theorem 3.1. Before beginning the proofs, let us first rephrase the data center optimization (3.1) as follows.

$$\begin{aligned} \text{minimize} \quad & \sum_{t=1}^{T} g_t(x_t) + \beta \sum_{t=1}^{T} y_t \\ \text{subject to} \quad & y_t \geq x_t - x_{t-1}, \quad x_t \geq 0, \quad y_t \geq 0. \end{aligned} \tag{3.7}$$

Next, we want to work with the dual of optimization (3.7). The Lagrangian (with respect to the first constraint) is

$$\begin{aligned} L(x, y, \nu) &= \sum_{t=1}^{T} g_t(x_t) + \beta \sum_{t=1}^{T} y_t + \sum_{t=1}^{T} \nu_t (x_t - x_{t-1} - y_t) \\ &= \sum_{t=1}^{T} (g_t(x_t) + (\beta - \nu_t) y_t) + \sum_{t=1}^{T-1} (\nu_t - \nu_{t+1}) x_t + \nu_T x_T - \nu_1 x_0 \end{aligned}$$

and the dual function is $D(\nu) = \inf_{x \geq 0, y \geq 0} L(x, y, \nu)$. Since we are interested in the maximum of $D(\nu)$ in the dual problem, we need only to consider the case $\beta - \nu_t \geq 0$. Then the dual function becomes

$$\begin{aligned} D(\nu) &= \inf_{x \geq 0} \left( \sum_{t=1}^{T} g_t(x_t) + \sum_{t=1}^{T-1} (\nu_t - \nu_{t+1}) x_t + \nu_T x_T - \nu_1 x_0 \right) \\ &= - \sum_{t=1}^{T-1} g_t^*(\nu_{t+1} - \nu_t) - g_T^*(-\nu_T) - \nu_1 x_0. \end{aligned}$$

where $g_t^*(\cdot)$ is the convex conjugate of function $g_t(\cdot)$ (enforcing $g_t(x_t) = \infty$ when $x_t < 0$). Therefore

the corresponding dual problem of (3.7) is

$$\begin{aligned} \max & -\sum_{t=1}^{T-1} g_t^*(\nu_{t+1}-\nu_t) - g_T^*(-\nu_T) - \nu_1 x_0 \\ \text{subject to } & 0 \le \nu_t \le \beta, \end{aligned} \tag{3.8}$$

where the complementary slackness conditions are

$$\nu_t(x_t - x_{t-1} - y_t) = 0 \tag{3.9}$$

$$(\beta - \nu_t) y_t = 0, \tag{3.10}$$

and the feasibility condition is $x_t \ge 0$, $y_t \ge 0$, $y_t \ge x_t - x_{t-1}$ and $0 \le \nu_t \le \beta$.

Using the above, we now observe a relationship between the data center optimization in (3.7) and the upper and lower bounds, i.e., optimizations (3.6) and (3.5). Specifically, if $\nu_{\tau+1} = 0$ in a solution of optimization (3.8), then $\nu_1, \ldots, \nu_\tau$ is a solution to:

$$\begin{aligned} \max & -\sum_{t=1}^{\tau-1} g_t^*(\nu_{t+1}-\nu_t) - g_\tau^*(-\nu_\tau) - \nu_1 x_0 \\ \text{subject to } & 0 \le \nu_t \le \beta, \end{aligned} \tag{3.11}$$

which is identical to the dual problem of optimization (3.5). Thus, the corresponding $x_1, \ldots, x_\tau$ is a solution to optimization (3.5). On the other hand, if $\nu_{\tau+1} = \beta$ in a solution of optimization (3.8), then $\nu_1, \ldots, \nu_\tau$ is a solution to:

$$\begin{aligned} \max & -\sum_{t=1}^{\tau-1} g_t^*(\nu_{t+1}-\nu_t) - g_\tau^*(\beta-\nu_\tau) - \nu_1 x_0 \\ \text{subject to } & 0 \le \nu_t \le \beta. \end{aligned} \tag{3.12}$$

Let $\nu'_t = \beta - \nu_t$, which makes (3.12) become

$$\begin{aligned} \max & -\sum_{t=1}^{\tau-1} g_t^*(\nu'_t-\nu'_{t+1}) - g_\tau^*(\nu'_\tau) + \nu'_1 x_0 - \beta x_0 \\ \text{subject to } & 0 \le \nu'_t \le \beta. \end{aligned} \tag{3.13}$$

It is easy to check that the corresponding $x_1, \ldots, x_\tau$ is a solution to optimization (3.6).

We require some notation and two technical lemmas before moving to the proofs of Lemma 3.1 and Theorem 3.1. Denote $h_{i,j}(x; x_S; x_E) = \sum_{t=i}^{j} g_t(x_t) + \beta(x_i - x_S)^+ + \beta \sum_{t=i+1}^{j} (x_t - x_{t-1})^+ + \beta(x_E - x_j)^+$ for $j \ge i$, and $h'_{i,j}(x; x_S; x_E) = \sum_{t=i}^{j} g_t(x_t) + \beta(x_S - x_i)^+ + \beta \sum_{t=i+1}^{j} (x_{t-1} - x_t)^+ +$

$\beta(x_j - x_E)^+$ for $j \geq i$. Then the objective of (3.5) is $h_{1,\tau}(x; x_0; 0)$. Similarly, the objective of (3.6) is $h'_{1,\tau}(x; x_0; x_M)$ where

$$x_M = \max(\max_{\tau,t} x^U_{\tau,t}, \max_t x^*_t). \tag{3.14}$$

The first lemma is to connect $h_{i,j}$ and $h'_{i,j}$:

**Lemma 3.3.** *Given $x_S$ and $x_E$, any solution minimizing $h_{i,j}(x; x_S; x_E)$ also minimizes $h'_{i,j}(x; x_S; x_E)$ and vice versa.*

*Proof.*

$$\begin{aligned} & h_{i,j}(x; x_S; x_E) - h'_{i,j}(x; x_S; x_E) \\ =& \beta(x_i - x_S) + \beta \sum_{t=i+1}^{j} (x_t - x_{t-1}) + \beta(x_E - x_j) \\ =& \beta(x_E - x_S), \end{aligned}$$

which is a constant. Thus any minimizer of $h_{i,j}(x; x_S; x_E)$ or $h'_{i,j}(x; x_S; x_E)$ is also a minimizer of the other. □

**Lemma 3.4.** *Given $x_S$ and $x_E \leq \hat{x}_E$, let $X = (x_i, \ldots, x_j)$ minimize $h_{i,j}(x; x_S; x_E)$, then there exists an $\hat{X} = (\hat{x}_i, \ldots, \hat{x}_j)$ minimizing $h_{i,j}(x; x_S; \hat{x}_E)$ such that $X \leq \hat{X}$.*

*Proof.* If there is more than one solution minimizing $h_{i,j}(x; x_S; \hat{x}_E)$, let $\hat{X}$ be a solution with greatest $\hat{x}_j$. We will first argue that $x_j \leq \hat{x}_j$. By definition, we have $h_{i,j}(X; x_S; x_E) \leq h_{i,j}(\hat{X}; x_S; x_E)$ and $h_{i,j}(\hat{X}; x_S; \hat{x}_E) \leq h_{i,j}(X; x_S; \hat{x}_E)$. Note that if the second inequality is an equality, then $x_j \leq \hat{x}_j$. Otherwise, sum the two inequalities, to obtain the strict inequality

$$(x_E - x_j)^+ + (\hat{x}_E - \hat{x}_j)^+ < (\hat{x}_E - x_j)^+ + (x_E - \hat{x}_j)^+.$$

Since $x_E \leq \hat{x}_E$, we can conclude that $x_j < \hat{x}_j$. Therefore, we always have $x_j \leq \hat{x}_j$.

Next, recursively consider the subproblem $h_{i,j-1}(\cdot)$ with $x_E = x_j$ and $\hat{x}_E = \hat{x}_j$. The same argument as above yields that $x_{j-1} \leq \hat{x}_{j-1}$. This continues and we can conclude that $(x_i, \ldots, x_j) \leq (\hat{x}_i, \ldots, \hat{x}_j)$. □

Lemma 3.4 shows that the optimizations have a unique maximal solution, as follows.

**Corollary 3.1.** *If there is more than one solution to optimization problem (3.1), (3.4), (3.5) or (3.6), there exists a maximum solution which is not less than others element-wise.*

*Proof.* Let us consider problem (3.1) with multiple solutions and prove the claim by induction. Assume that $x^*$ is a solution with the first $\tau$ ($\tau \geq 1$) entries not less than those in other solutions,

but $x^*_{\tau+1} < \hat{x}^*_{\tau+1}$ where $\hat{x}^*$ is a solution with the greatest $(\tau+1)$th entry. Since $(x^*_1, \dots, x^*_\tau)$ minimizes $h_{1,\tau}(x; x_0; x^*_{\tau+1})$, Lemma 3.4 implies there exists a solution $(\bar{x}^*_1, \dots, \bar{x}^*_\tau)$ minimizing $h_{1,\tau}(x; x_0; \hat{x}^*_{\tau+1})$ which is not less than $(x^*_1, \dots, x^*_\tau)$. Thus we can replace $(\hat{x}^*_1, \dots, \hat{x}^*_\tau)$ in $\hat{x}^*$ by $(\bar{x}^*_1, \dots, \bar{x}^*_\tau)$ to get a solution with the first $\tau+1$ entries not less than other solutions. The proof for optimization problem (3.4), (3.5) and (3.6) are similar and thus omitted. □

We now complete the proofs of Lemma 3.1 and Theorem 3.1.

*Proof of Lemma 3.1.* Let $X^L_\tau = (x^L_{\tau,1}, x^L_{\tau,2}, \dots, x^L_{\tau,\tau})$ be the solution of optimization (3.5) at time $\tau$ and define $X^U_\tau$ symmetrically for (3.6). Also, let $X^*_\tau = (x^*_1, \dots, x^*_\tau)$ be the first $\tau$ entries of the offline solution to optimization (3.1).

We know that $X^L_\tau$ minimizes $h_{1,\tau}(x; x_0; 0)$ and $X^*_\tau$ minimizes $h_{1,\tau}(x; x_0; x^*_{\tau+1})$. Since $x^*_{\tau+1} \geq 0$, Lemma 3.4 implies $X^*_\tau \geq X^L_\tau$, and thus, in particular, the last entry satisfies $x^*_\tau \geq x^L_{\tau,\tau}$.

Symmetrically, $X^U_\tau$ minimizes $h'_{1,\tau}(x; x_0; x_M)$ where $x_M$ satisfies (3.14). By Lemma 3.3, $X^U_\tau$ also minimizes $h_{1,\tau}(x; x_0; x_M)$. Since $x^*_{\tau+1} \leq x_M$, Lemma 3.4 implies that $X^*_\tau \leq X^U_\tau$, and thus $x^*_\tau \leq x^U_{\tau,\tau}$. □

*Proof of Theorem 3.1.* As a result of Lemma 3.1, we know that $x^*_\tau \in [x^L_\tau, x^U_\tau]$ for all $\tau$. Further, if $x^*_\tau > x^*_{\tau+1}$, by the complementary slackness condition (3.9), we have that $\nu_{\tau+1} = 0$. Thus, in this case, $x^*_\tau$ solves optimization (3.5) for the lower bound, i.e., $x^*_\tau = x^L_\tau$. Symmetrically, if $x^*_\tau < x^*_{\tau+1}$, we have that complementary slackness condition (3.10) gives $\nu_{\tau+1} = \beta$ and so $x^*_\tau$ solves optimization (3.6) for the upper bound, i.e., $x^*_\tau = x^U_\tau$. Thus, whenever $x^*_\tau$ is increasing/decreasing it must match the upper/lower bound, respectively. □

## Appendix 3.B Analysis of lazy capacity provisioning, LCP($w$)

In this section we will prove Lemma 3.2 and Theorem 3.2.

*Proof of Lemma 3.2.* First, we prove that $x^{L,w}_\tau \leq x^*_\tau$. By definition, $x^{L,w}_\tau = x^L_{\tau+w,\tau}$, and so it belongs to a solution minimizing $h_{1,\tau+w}(x; x_0; 0)$. Further, we can view the optimal $x^*_\tau$ as an entry in a solution minimizing $h_{1,\tau+w}(x; x_0; x^*_{\tau+w+1})$. From these two representations, we can apply Lemma 3.4, to conclude that $x^{L,w}_\tau \leq x^*_\tau$.

Next, we prove that $x^L_\tau \leq x^{L,w}_\tau$. To see this we notice that $x^L_\tau$ is the last entry in a solution minimizing $h_{1,\tau}(x; x_0; 0)$. And we can view $x^{L,w}_\tau$ as an entry in a solution minimizing $h_{1,\tau}(x; x_0; x^L_{\tau+w,\tau+1})$ where $x^L_{\tau+w,\tau+1} \geq 0$. Based on Lemma 3.4 we get $x^L_\tau \leq x^{L,w}_\tau$.

The proof that $x^*_\tau \leq x^{U,w}_\tau \leq x^U_\tau$ is symmetric. □

From the above lemma, we immediately obtain an extension of the characterization of the offline optimum.

**Corollary 3.2.** *The optimal solution of the data center optimization (3.1) satisfies the following backwards recurrence relation*

$$x^*_\tau = \begin{cases} 0, & \tau > T; \\ (x^*_{\tau+1})^{x^{U,w}_\tau}_{x^{L,w}_\tau}, & \tau \leq T. \end{cases}$$

Moving to the proof of Theorem 3.2, the first step is to use the above lemmas to characterize the relationship between $x^{LCP(w)}_\tau$ and $x^*_\tau$. Note that $x^{LCP(w)}_0 = x^*_0 = x^{LCP(w)}_{T+1} = x^*_{T+1} = 0$.

**Lemma 3.5.** *Consider the timeslots $0 = t_0 < t_1 < \ldots < t_m = T$ such that $x^{LCP(w)}_{t_i} = x^*_{t_i}$. Then, during each segment $(t_{i-1}, t_i)$, either*

*(i) $x^{LCP(w)}_t > x^*_t$ and both $x^{LCP(w)}_t$ and $x^*_t$ are non-increasing for all $t \in (t_{i-1}, t_i)$, or*

*(ii) $x^{LCP(w)}_t < x^*_t$ and both $x^{LCP(w)}_t$ and $x^*_t$ are non-decreasing for all $t \in (t_{i-1}, t_i)$.*

*Proof.* The result follows from the characterization of the offline optimal solution in Corollary 3.2 and the definition of LCP($w$). Given that both the offline optimal solution and LCP($w$) are non-constant only for timeslots when they are equal to either $x^{U,w}_t$ or $x^{L,w}_t$, we know that at any time $t_i$ where $x^{LCP(w)}_{t_i} = x^*_{t_i}$ and $x^{LCP(w)}_{t_i+1} \neq x^*_{t_i+1}$, we must have that both $x^{LCP(w)}_{t_i}$ and $x^*_{t_i}$ are equal to either $x^{U,w}_{t_i}$ or $x^{L,w}_{t_i}$.

Now we must consider two cases. First, consider the case that $x^{LCP(w)}_{t_i+1} > x^*_{t_i+1}$. It is easy to see that $x^{LCP(w)}_{t_i+1}$ does not match the lower bound since $x^*_{t_i+1}$ is not less than the lower bound. Thus $x^{LCP(w)}_{t_i} \geq x^{LCP(w)}_{t_i+1}$ since, by definition, LCP($w$) will never choose to increase the number of servers it uses unless it matches the lower bound. Consequently, it must be that $x^*_{t_i} = x^{LCP(w)}_{t_i} \geq x^{LCP(w)}_{t_i+1} > x^*_{t_i+1}$. Since $x^*$ is decreasing, both $x^{LCP(w)}_{t_i}$ and $x^*_{t_i}$ match the lower bound. Further, the next time, $t_{i+1}$, when the optimal solution and LCP($w$) match is the next time either the number of servers in LCP($w$) matches the lower bound $x^{L,w}_t$ or the next time the number of servers in the optimal solution matches the upper bound $x^{U,w}_t$. Thus, until that point, LCP($w$) cannot increase the number of servers (since this happens only when it matches the lower bound) and the optimal solution cannot increase the number of servers (since this happens only when it matches the upper bound). This completes the proof of part (i) of the lemma. The proof of part (ii) is symmetric. □

Given Lemma 3.5, we bound the switching cost of LCP($w$).

**Lemma 3.6.** $cost_s(X^{LCP(w)}) = cost_s(X^*)$.

*Proof.* Consider the sequence of times $0 = t_0 < t_1 < \ldots < t_m = T$ such that $x^{LCP(w)}_{t_i} = x^*_{t_i}$ identified in Lemma 3.5. Then, each segment $(t_{i-1}, t_i)$ starts and ends with the same number of servers being used under both LCP($w$) and the optimal solution. Additionally, the number of servers is monotone for both LCP($w$) and the optimal solution, thus the switching cost incurred by LCP($w$) and the optimal solution during each segment is the same. □

Next, we bound the operating cost of LCP($w$).

**Lemma 3.7.** $cost_o(X^{LCP(w)}) \leq cost_o(X^*) + \beta \sum_{t=1}^{T} |x_t^* - x_{t-1}^*|$.

*Proof.* Consider the sequence of times $0 = t_0 < t_1 < \ldots < t_m = T$ such that $x_{t_i}^{LCP(w)} = x_{t_i}^*$ identified in Lemma 3.5, and consider specifically one of these intervals $(t_{i-1}, t_i)$ such that $x_{t_{i-1}}^{LCP(w)} = x_{t_{i-1}}^*$, $x_{t_i}^{LCP(w)} = x_{t_i}^*$.

There are two cases in the proof: (i) $x_t^{LCP(w)} > x_\tau^*$ for all $\tau \in (t_{i-1}, t_i)$ and (ii) $x_t^{LCP(w)} < x_\tau^*$ for all $t \in (t_{i-1}, t_i)$.

We handle *case (i)* first. Our goal is to prove that

$$\sum_{t=t_{i-1}+1}^{t_i} g_t(x_t^{LCP(w)}) \leq \sum_{t=t_{i-1}+1}^{t_i} g_t(x_t^*) + \beta |x_{t_{i-1}}^* - x_{t_i}^*|. \tag{3.15}$$

Define $X_\tau = (x_{\tau,1}, \ldots, x_{\tau,\tau-1}, x_\tau^{LCP(w)})$ where $(x_{\tau,1}, \ldots, x_{\tau,\tau-1})$ minimizes $h'_{1,\tau-1}(x; x_0; x_\tau^{LCP(w)})$.

Additionally, define $X'_\tau = (x'_{\tau,1}, \ldots, x'_{\tau,\tau})$ as the solution minimizing $h'_{1,\tau}(x; x_0; x_\tau^{LCP(w)})$; by Lemma 3.3 this also minimizes $h_{1,\tau}(x; x_0; x_\tau^{LCP(w)})$.

We first argue that $x'_{\tau,\tau} = x_\tau^{LCP(w)}$ via a proof by contradiction. Note that if $x'_{\tau,\tau} > x_\tau^{LCP(w)}$, based on similar argument in the proof of Theorem 3.1, we have $x'_{\tau,\tau} = x_\tau^L$, which contradicts the fact that $x'_{\tau,\tau} > x_\tau^{LCP(w)} \geq x_\tau^L$. Second, if $x'_{\tau,\tau} < x_\tau^{LCP(w)}$, then we can follow a symmetric argument to arrive at a contradiction. Thus $x'_{\tau,\tau} = x_\tau^{LCP(w)}$.

Consequently, $x'_{\tau,t} = x_{\tau,t}$ for all $t \in [1, \tau]$ and we get

$$h'_{1,\tau}(X'_\tau; x_0; x_M) = h'_{1,\tau}(X_\tau; x_0; x_M) \tag{3.16}$$

Next, let us consider $X_{\tau+1} = (x_{\tau+1,1}, \ldots, x_{\tau+1,\tau}, x_{\tau+1}^{LCP(w)})$ where $(x_{\tau+1,1}, \ldots, x_{\tau+1,\tau})$ minimizes $h'_{1,\tau}(x; x_0; x_{\tau+1}^{LCP(w)})$. Recalling $x_t^{LCP(w)}$ is non-increasing in case (i) by Lemma 3.5, we have $X_{\tau+1} \leq X'_\tau$ by Lemma 3.4. In particular, $x_{\tau+1,\tau} \leq x_\tau^{LCP(w)}$. Thus

$$\begin{aligned} h'_{1,\tau+1}(X_{\tau+1}; x_0; x_M) \geq & h'_{1,\tau}((x_{\tau+1,1}, \ldots, x_{\tau+1,\tau}); x_0; x_M) + g_{\tau+1}(x_{\tau+1}^{LCP(w)}) \\ = & h'_{1,\tau}((x_{\tau+1,1}, \ldots, x_{\tau+1,\tau}); x_0; x_\tau^{LCP(w)}) + g_{\tau+1}(x_{\tau+1}^{LCP(w)}) \end{aligned} \tag{3.17}$$

By definition of $X'_\tau$, we get

$$\begin{aligned} & h'_{1,\tau}((x_{\tau+1,1}, \ldots, x_{\tau+1,\tau}); x_0; x_\tau^{LCP(w)}) \\ \geq & h'_{1,\tau}(X'_\tau; x_0; x_\tau^{LCP(w)}) \geq h'_{1,\tau}(X'_\tau; x_0; x_M) \end{aligned} \tag{3.18}$$

Combining equations (3.16), (3.17) and (3.18), we obtain

$$h'_{1,\tau+1}(X_{\tau+1}; x_0; x_M) \geq h'_{1,\tau}(X_\tau; x_0; x_M) + g_{\tau+1}(x_{\tau+1}^{LCP(w)}).$$

By summing this equality for $\tau \in [t_{i-1}, t_i)$, we have

$$\sum_{t=t_{i-1}+1}^{t_i} g_t(x_t^{LCP(w)}) \leq h'_{1,t_i}(X_{t_i}; x_0; x_M) - h'_{1,t_{i-1}}(X_{t_{i-1}}; x_0; x_M).$$

Since $x_{t_{i-1}}^{LCP(w)} = x_{t_{i-1}}^*$, $x_{t_i}^{LCP(w)} = x_{t_i}^*$, we know that both $X_{t_{i-1}}$ and $X_{t_i}$ are prefixes[2] of the offline solution $x^*$, thus $X_{t_{i-1}}$ is the prefix of $X_{t_i}$. Expanding out $h'(\cdot)$ in the above inequality gives (3.15), which completes case (i).

In *case (ii)*, i.e., segments where $x_t^{LCP(w)} < x_\tau^*$ for all $t \in (t_{i-1}, t_i)$, a parallel argument shows that (3.15) again holds.

To complete the proof we combine the results from case (i) and case (ii), summing equation (3.15) over all segments (and the additional times when $x_t^{LCP(w)} = x_t^*$). □

We can now prove the competitive ratio in Theorem 3.2.

**Lemma 3.8.** *$cost(X^{LCP(w)}) \leq cost(X^*) + 2cost_s(X^*)$. Thus, LCP(w) is 3-competitive for the data center optimization (3.1).*

*Proof.* Combining Lemma 3.7 and Lemma 3.6 gives that $cost(X^{LPC(w)}) \leq cost(X^*) + \beta|x_t^* - x_{t-1}^*|$. Note that, because both LCP($w$) and the optimal solution start and end with zero servers on, we have $\sum_{t=1}^{T} |x_t^* - x_{t-1}^*| = 2\sum_{t=1}^{T}(x_t^* - x_{t-1}^*)^+$, which completes the proof. □

All that remains for the proof of Theorem 3.2 is to prove that the bound of 3 on the competitive ratio is tight.

**Lemma 3.9.** *The competitive ratio of LCP(w) is at least 3.*

*Proof.* The following is a family of instances parameterized by $n$ and $m \geq 2$, whose competitive ratios approach the bound of 3. The operating cost for each server is defined as $f(z) = z^m + f_0$ for $0 \leq z \leq 1$ and $f(z) = \infty$ otherwise, whence $g_t(x_t) = x_t f(\lambda_t / x_t)$. The switching cost is $\beta = 0.5$. Let $\delta \in (1, 1.5)$ be such that $n = \log_\delta \frac{1}{\delta - 1}$. The arrival rate at time $i$ is $\lambda_i = \delta^{i-1}$ for $1 \leq i \leq n$, and $\lambda_i = 0$ for $n < i \leq T$, where $T > \beta/f_0 + n$ with $f_0 = \frac{\beta(\delta^m - 1)}{n(\delta^{mn} - 1)}$.

For the offline optimization, denote the solution by vector $x^*$. First note that $x_i^* = x_n^*$ for $i \in [1, n]$ since $x_i^*$ is non-decreasing for $i \in [1, n]$ and, for the above $f$,

$$\frac{d}{dx}[xf(\lambda_i/x)] < 0 \text{ for } x \in [\lambda_i, x_n^*]. \tag{3.19}$$

[2]That is, $X_{t_{i-1}}$ is the first $t_{i-1}$ components of $x^*$ and $X_{t_i}$ is the first $t_i$ components of $x^*$.

Moreover, $x_i^* = 0$ for $i \in [n+1, T]$. Hence the minimum cost is $\sum_{i=1}^{n} \frac{\lambda_i^m}{(x_n^*)^{m-1}} + (nf_0 + \beta)x_n^*$. Then by the first order (stationarity) condition we get

$$(x_n^*)^{-m} = \frac{(nf_0 + \beta)(\delta^m - 1)}{(m-1)(\delta^{mn} - 1)}, \tag{3.20}$$

which is smaller than $\lambda_n^{-m}$ because $nf_0 + \beta \leq 1$, $m - 1 \geq 1$ and $\delta > 1$. Thus it is feasible ($f(\lambda_i/x_n^*)$ is finite). The cost for the offline optimal solution is then

$$C^* = \frac{m}{m-1}(nf_0 + \beta)x_n^*.$$

We consider $LCP(w)$ with $w = 0$ before considering the general case. Let $C_{[i,j]}$ denote the cost of LCP(0) on $[i, j]$.

For the online algorithm LCP(0), denote the result by vector $\hat{x}$. We know $\hat{x}_i$ is actually matching $x_i^L$ in $[1, n]$ ($x_i^U$ is not less than $x_i^* = x_n^*$), thus $\hat{x}_i$ is non-decreasing for $i \in [1, n]$ and $\hat{x}_n = x_n^*$. By the same argument as for $x_n^*$, we have

$$(\hat{x}_\tau)^{-m} = \frac{(\tau f_0 + \beta)(\delta^m - 1)}{(m-1)(\delta^{m\tau} - 1)} \tag{3.21}$$

Thus the cost for LCP(0) in $[1, n]$ is

$$\begin{aligned}
C_{[1,n]} &= \sum_{\tau=1}^{n} \left( \frac{\lambda_\tau^m}{(\hat{x}_\tau)^{m-1}} + f_0 \hat{x}_\tau \right) + \beta x_n^* \\
&> \sum_{\tau=1}^{n} \delta^{m(\tau-1)} \left( \frac{(\tau f_0 + \beta)(\delta^m - 1)}{(m-1)(\delta^{m\tau} - 1)} \right)^{\frac{m-1}{m}} + \beta x_n^* \\
&> \left( \frac{\beta(\delta^m - 1)}{m-1} \right)^{\frac{m-1}{m}} \sum_{\tau=1}^{n} \delta^{\tau - m} + \beta x_n^* \\
&= \left( \frac{\beta(\delta^m - 1)}{m-1} \right)^{\frac{m-1}{m}} \frac{\delta^n - 1}{(\delta - 1)\delta^{m-1}} + \beta x_n^*
\end{aligned}$$

Thus by (3.20)

$$\begin{aligned}
\frac{C_{[1,n]}}{x_n^*} &> \frac{\delta^n - 1}{(\delta^{mn} - 1)^{1/m}} \cdot \frac{\beta(\delta^m - 1)}{(m-1)(\delta - 1)\delta^{m-1}} + \beta \\
&> \frac{\delta^n - 1}{\delta^n} \cdot \frac{\beta(\delta^m - 1)}{(m-1)(\delta - 1)\delta^{m-1}} + \beta
\end{aligned}$$

As $n \to \infty$, both $\delta \to 1$ and $(\delta^n - 1)/\delta^n \to 1$, since $n = \log_\delta \frac{1}{\delta - 1}$. Since $m$ is independent of $\delta$,

L'Hospital's Law gives

$$\begin{aligned}\lim_{n\to\infty}\frac{C_{[1,n]}}{x_n^*} &\geq \lim_{\delta\to 1}\frac{\beta m\delta^{m-1}}{(m-1)(m\delta^{m-1}-(m-1)\delta^{m-2})}+\beta\\ &= \frac{m}{m-1}\beta+\beta\end{aligned}$$

Now let us calculate the cost for LCP(0) in $[n+1,T]$.

For $\tau > n$, by LCP(0), we know that $x_\tau$ will stay constant until it hits the upper bound. Let $X_\tau = \{x_{\tau,t}\}$ be the solution of optimization (3.6) in $[1,\tau]$ for any $\tau > n$. We now prove that for $\tau$ such that $(\tau-n)f_0 < \beta$, we have $x_{\tau,\tau} \geq x_n^*$, and thus $\hat{x}_\tau = x_n^*$.

Note that $x_{\tau,n} \geq x_n^*$ since $x_n^*$ belongs to the lower bound. Given $x_{\tau,n}$, we know that $x_{\tau,n+1},\ldots,x_{\tau,\tau}$ is the solution to the following problem:

$$\begin{aligned}\text{minimize}\quad & \sum_{t=n+1}^{\tau} f_0 X_{\tau,t} + \beta\sum_{t=n+1}^{\tau}(X_{\tau,t-1}-X_{\tau,t})^+\\ \text{subject to}\quad & X_{\tau,t}\geq 0\end{aligned}$$

Let $x_{\min} = \min\{x_{\tau,n},\ldots,x_{\tau,\tau}\}$. Then

$$\begin{aligned}& \sum_{t=n+1}^{\tau} f_0 x_{\tau,t} + \beta\sum_{t=n+1}^{\tau}(x_{\tau,t-1}-x_{\tau,t})^+\\ \geq\ & \sum_{t=n+1}^{\tau} f_0 x_{\min} + \beta(x_{\tau,n}-x_{\min})\\ \geq\ & (\tau-n)f_0 x_{\tau,n}\end{aligned}$$

The last inequality is obtained by substituting $\beta > (\tau-n)f_0$. We can see that $x_{\tau,i} = x_{\tau,n} (i \in [n+1,\tau])$ is a solution, thus $x_{\tau,\tau} \geq x_n^*$. (Recall that, if there are multiple solutions, we take the maximum one). Therefore, we have

$$C_{[n+1,\tau]} = (\tau-n)f_0 x_n^*.$$

Since $f_0 \to 0$ as $\delta \to 1$, we can find an $\tau$ so that $(\tau-n)f_0 \to \beta$, and thus

$$C_{[n+1,\tau]} \to \beta x_n^*. \tag{3.22}$$

By combining the cost for LCP(0) in $[1,n]$ and $[n+1,T]$, we have

$$C_{[1,T]}/C^* \geq \frac{\frac{m}{m-1}\beta+2\beta}{\frac{m}{m-1}(nf_0+\beta)} = \frac{3-2/m}{1+nf_0/\beta}.$$

Using the relationship between $n$, $f_0$ and $\delta$, we can choose a large enough $m$ and $n$ to make this

arbitrarily close to 3. This finishes the proof for LCP(0).

Now let us consider LCP($w$) for $w > 0$. Denote the solution of LCP($w$) by $\hat{x}'$. At time $\tau \in [1, n-w]$, LCP($w$) solves the same optimization problem as LCP(0) does at $\tau + w$. Thus $\hat{x}'_\tau = \hat{x}_{\tau+w}$ of LCP(0). Thus

$$\begin{aligned} C'_{[1,n-w]} &= \sum_{\tau=1}^{n-w} \left( \frac{\lambda_\tau^m}{(\hat{x}'_\tau)^{m-1}} + f_0 \hat{x}'_\tau \right) + \beta x_n^* \\ &= \sum_{\tau=1+w}^{n} \left( \frac{1}{\delta^{wm}} \frac{\lambda_\tau^m}{(\hat{x}_\tau)^{m-1}} + f_0 \hat{x}_\tau \right) + \beta x_n^* \\ &> \frac{1}{\delta^{wm}} C_{[1+w,n]} \end{aligned}$$

By pushing $n \to \infty$, and hence $\delta \to 1$, we have $C'_{[1,n-w]}/C_{[1,n]} \to 1$.

And for $\tau > n+1$, if $f_0(\tau - n) < \beta$, then $C'_{[n+1,\tau-w]} = (\tau - w - n) f_0 x_n^*$. By pushing $\delta \to 1$, we can find a $\tau$ such that $C'_{[n+1,\tau-w]} \to \beta x_n^*$, the same as (3.22). Therefore, as $\delta \to 1$, we have $C'_{[1,T]}/C_{[1,T]} \to 1$, thus the supremum over arbitrarily large $m \geq 2$ and arbitrarily small $\delta > 1$ of the competitive ratio of LCP($w$) on this family is also 3. □

Finally, the following lemma ensures that the optimizations solved by LCP($w$) at each timeslot $\tau$ remain small.

**Lemma 3.10.** *If there exists an index $\tau \in [1, t-1]$ such that $x^U_{t,\tau+1} < x^U_{t,\tau}$ or $x^L_{t,\tau+1} > x^L_{t,\tau}$, then $(x^U_{t,1}, \ldots, x^U_{t,\tau}) = (x^L_{t,1}, \ldots, x^L_{t,\tau})$. No matter what the future functions $g_i(\cdot)$ are, solving either (3.5) or (3.6) in $[1, t']$ for $t' > t$ is equivalent to solving two optimizations: one over $[1, \tau]$ with initial condition $x_0$ and final condition $x^U_{t,\tau}$ and the second over $[\tau + 1, t']$ with initial condition $x^U_{t,\tau}$.*

*Proof.* Consider the case $x^L_{t,\tau+1} > x^L_{t,\tau}$. By the complementary slackness conditions (3.10), the corresponding dual variable $\nu_{\tau+1} = \beta$. Based on the argument in the proof of Theorem 3.1, we know the dual problem up to time $\tau$ is (3.12), which is identical to (3.13), the dual problem of (3.6). Since the optimal $x_i$ depends only on $\nu_i$ and $\nu_{i+1}$, the optimal $x_i$ for $i \leq \tau$ are completely determined by $\nu_i$ for $i \in [1, \tau+1]$, with $\nu_{\tau+1} = \beta$. Hence $(x^L_{t,1}, \ldots, x^L_{t,\tau}) = (x^U_{t,1}, \ldots, x^U_{t,\tau})$. The proof for the case $x^U_{t,\tau+1} < x^U_{t,\tau}$ is symmetric.

Notice that $(x^U_{t,1}, \ldots, x^U_{t,t})$ minimizes $h'_{1,t}(x; x_0; x_M)$ and thus $h_{1,t}(x; x_0; x_M)$ based on Lemma 3.3, $(x^L_{t,1}, \ldots, x^L_{t,t})$ minimizes $h_{1,t}(x; x_0; 0)$. No matter what the future functions $g_i(\cdot)$ are, the first $t$ entries of its solution must minimize $h_{1,t}(x; x_0; x_{t+1})$ for some $x_{t+1} \in [0, x_M]$ by the maximality of $x_M$. Based on Lemma 3.4, the first $t$ entries are bounded by $(x^U_{t,1}, \ldots, x^U_{t,t})$ and $(x^L_{t,1}, \ldots, x^L_{t,t})$. However, we have seen that $(x^U_{t,1}, \ldots, x^U_{t,\tau}) = (x^L_{t,1}, \ldots, x^L_{t,\tau})$, thus the first $\tau$ entries of its solution are equal to $(x^U_{t,1}, \ldots, x^U_{t,\tau})$ no matter what the future is. □

# Chapter 4

# Cost-Effective Geographical Load Balancing

As shown in the previous chapters, energy consumption of data centers is a major concern to both operators and society. Electricity for Internet-scale systems costs millions of dollars per month [102] and, though IT uses only a small percentage of electricity today, the growth of electricity in IT exceeds nearly all sectors of the economy. For these reasons, and more, IT must play its part in reducing our dependence on fossil fuels. This can be achieved by using renewable energy to power data centers. Already, data centers are starting to be powered by a greener portfolio of energy [114, 91, 93]. However, achieving a goal of powering data centers *entirely* with renewable energy is a significant challenge due to the intermittency and unpredictability of renewable energy. Most studies of powering data centers *entirely* with renewable energy have focused on powering individual data centers, e.g., [50, 49]. These have shown that it is challenging to power a data center using only local wind and solar energy without large-scale storage, due to the intermittency and unpredictability of these sources.

The goal of this chapter is twofold: (i) to illustrate that the geographical diversity of Internet-scale services can significantly improve the efficiency of the usage of renewable energy, and (ii) to develop online algorithms that can realize this potential.

Many papers have illustrated the potential for using "geographical load balancing" (GLB) to exploit the diversity of Internet-scale service and provide significant *cost savings* for data centers; see [102, 98, 104, 106]. The goal of this chapter is different. It is to explore the environmental impact of GLB within Internet-scale systems. In particular, using GLB to reduce *cost* can actually increase total energy usage: reducing the average price of energy shifts the economic balance away from energy-saving measures. However, more positively, if data centers have local renewable energy available, GLB provides a huge opportunity by allowing for "follow the renewables" routing.

Research is only beginning to quantify the benefits of this approach, e.g., [88] and [87]. Many questions remain. For example: Does "follow the renewables" routing make it possible to attain

"net-zero" Internet-scale services? What is the optimal mix of renewable energy sources (e.g., wind and solar) for an Internet-scale service? To address these questions, we perform a numerical study using real-world traces for workloads, electricity prices, renewable availability, data center locations, etc. Surprisingly, our study shows that wind energy is significantly more valuable than solar energy for "follow the renewables" routing. Commonly, solar is assumed to be more valuable given the match between the peak traffic period and the peak period for solar energy. Wind energy lacks this correlation, but also has little correlation across locations and is available during both night and day; thus the aggregate wind energy over many locations exhibits much less variation than that of solar energy [8].

Our numerical results suggest that using GLB for "follow the renewables" routing can provide significant environmental benefits. However, achieving this is a challenging algorithmic task. The benefits come from dynamically adjusting the routing and service capacity at each location, but the latter incurs a significant "switching cost" in the form of latency, energy consumption, and/or wear-and-tear. Further, predictions of the future workload, renewable availability, and electricity price are inaccurate beyond the short term. Thus online algorithms are required for GLB.

Although the distant future cannot be known, it is often possible to estimate loads a little in the future [123]. These predictions can be used by algorithms such as receding horizon control (RHC), also known as model predictive control, to perform geographical load balancing. RHC is commonly proposed to control data centers [116, 78] and has a long history in control theory [79]. In RHC, an estimate of the near future is used to design a tentative control trajectory; only the first step of this trajectory is implemented and, in the next time step, the process repeats.

Due to its use in systems today, we begin in Section 4.2 by analyzing the performance of RHC applied to the model of Section 4.1. In particular, we study its competitive ratio: the worst-case ratio of the cost of using RHC to the cost of using optimal provisioning based on perfect future knowledge. We prove that RHC does work well in some settings, e.g., in a homogeneous setting (where all servers are equally able to serve every request) RHC is $1 + O(1/w)$-competitive, where $w$ is the size of the prediction window. However, RHC can perform badly for the heterogeneous settings needed for geographical load balancing. In general, RHC is $1 + \Omega(\beta/e_0)$-competitive, where $\beta$ measures the switching cost and $e_0$ is the cost of running an idle server. This can be large and, surprisingly, does not depend on $w$. That is, the worst-case bound on RHC does not improve as the prediction window grows.

Motivated by the weakness of RHC in the general context of geographical load balancing, we design a new algorithm in Section 4.2 called averaging fixed horizon control (AFHC). AFHC works by taking the average of $w + 1$ fixed horizon control (FHC) algorithms. Alone, each FHC algorithm seems much worse than RHC, but by combining them AFHC achieves a competitive ratio of $1 + O(1/w)$, superior to that of RHC. We evaluate these algorithms in Section 4.3 under real data

center workloads, and show that the improvement in worst-case performance comes at no cost to the average-case performance.

Note that the analysis of RHC and AFHC applies to a very general model. It allows heterogeneity among both the jobs and the servers, whereas systems studied analytically typically have homogeneous servers [46, 36] or disjoint collections thereof [63].

## 4.1 Model and notation

Our focus is on understanding how to dynamically provision the (active) service capacity in geographically diverse data centers serving requests from different regions so as to minimize the "cost" of the system, which may include both energy and quality of service. In this section, we introduce a simple but general model for this setting. Note that the model generalizes most recent analytic studies of both dynamic resizing within a local data center and geographical load balancing among geographically distributed data centers, e.g., including the previous chapter and [88, 53, 102].

### 4.1.1 The workload

Similar to the workload model in Chapter 3, we consider a discrete-time model whose timeslot matches the timescale at which routing decisions and capacity provisioning decisions can be updated. There is a (possibly long) interval of interest $t \in \{1, \dots, T\}$. There are $J$ geographically concentrated sources of requests, and the mean arrival rate at time $t$ is denoted by $\lambda_t = (\lambda_{t,j})_{j \in \{1,\dots,J\}}$, where $\lambda_{t,j}$ is the mean request rate from source $j$ at time $t$. We set $\lambda_t = 0$ for $t < 1$ and $t > T$. In a real system, $T$ could be a year, a timeslot could be 10 minutes.

### 4.1.2 The Internet-scale system

We model an Internet-scale system as a collection of $S$ geographically diverse data centers, where data center $s \in S$ is modeled as a collection of $M_s$ homogeneous servers.[1] We seek the values of two key GLB parameters:

(i) $\lambda_{t,j,s}$, the amount of traffic routed from source $j$ to data center $s$ at time $t$, such that $\sum_{s=1}^{S} \lambda_{t,j,s} = \lambda_{t,j}$.

(ii) $x_t = (x_{t,s})_{s \in \{1,\dots,S\}}$, where $x_{t,s} \in \{0, \dots, M_s\}$ is the number of active servers at data center $s$ at time $t$.

The objective is to choose $\lambda_{t,j,s}$ and $x_t$ to minimize the "cost" of the system, which can be decomposed into two components:

[1] Note that a heterogeneous data center can simply be viewed as multiple data centers, each having homogeneous servers.

(i) The *operating cost* incurred by using active servers. It includes both the delay cost (revenue loss) which depends on the dispatching rule through network delays and the load at each data center, and also the energy cost of the active servers at each data center with particular load.

(ii) The *switching cost* incurred by toggling servers into and out of a power-saving mode between timeslots (including the delay, migration, and wear-and-tear costs).

We describe each of these in detail below.

**Operating cost**

The operating cost is the sum of the delay cost and the energy cost. Each is described below.

*Delay cost:* The delay cost captures the lost revenue incurred because of the delay experienced by the requests. To model this, we define $r_t(d)$ as the lost revenue associated with a job experiencing delay $d$ at time $t$, which is an increasing and convex function. The delay has two components: the network delay experienced while the request is outside of the data center and the queueing delay experienced while the request is at the data center.

We model the *network delays* by a fixed delay $\delta_{t,j,s}$ experienced by a request from source $j$ to data center $s$ during timeslot $t$. We make no requirements on the structure of the $\delta_{t,j,s}$. We assume that these delays are known within the prediction window $w$.

To model the *queueing delay*, we let $q_s(x_{t,s}, \sum_j \lambda_{t,j,s})$ denote the queueing delay at data center $s$ given $x_{t,s}$ active servers and an arrival rate of $\sum_j \lambda_{t,j,s}$. Further, for stability, we must have that $\sum_j \lambda_{t,j,s} < x_{t,s}\mu_s$, where $\mu_s$ is the service rate of a server at data center $s$. Thus, we define $q_s(x_{t,s}, \sum_j \lambda_{t,j,s}) = \infty$ for $\sum_j \lambda_{t,j,s} \geq x_{t,s}\mu_s$.

Combining the above gives the following model for the total delay cost $\mathcal{D}_{t,s}$ at data center $s$ during timeslot $t$:

$$\mathcal{D}_{t,s} = \sum_{j=1}^{J} \lambda_{t,j,s} r_t \left( q_s \left( x_{t,s}, \sum\nolimits_{j'} \lambda_{t,j',s} \right) + \delta_{t,j,s} \right). \tag{4.1}$$

We assume that $\mathcal{D}_{t,s}$ is jointly convex in $x_{t,s}$ and $\lambda_{t,j,s}$. Note that this assumption is satisfied by most standard queueing formulae, e.g., the mean delay under M/GI/1 processor sharing (PS) queue and the 95th percentile of delay under the M/M/1.

*Energy cost:* To capture the geographic diversity and variation over time of energy costs, we let $f_{t,s}(x_{t,s}, \sum_j \lambda_{t,j,s})$ denote the energy cost for data center $s$ during timeslot $t$ given $x_{t,s}$ active servers and arrival rate $\sum_j \lambda_{t,j,s}$. For every fixed $t$, we assume that $f_{t,s}(x_{t,s}, \sum_j \lambda_{t,j,s})$ is jointly convex in $x_{t,s}$ and $\lambda_{t,j,s}$. This formulation is quite general, and captures, for example, the common charging plan of a fixed price per kWh plus an additional "demand charge" for the peak of the average power used over a sliding 15 minute window [97]. Additionally, it can capture a wide range of models for server power consumption, e.g., energy costs as an affine function of the load, see [45], or as a

polynomial function of the speed, see [120] and Chapter 2. One important property of $f_{t,s}$ for our results is $e_{0,s}$, the minimum cost per timeslot for an active server of type $s$, i.e., $f_{t,s}(x_{t,s}, \cdot) \geq e_{0,s}x_{t,s}$.

The total energy cost of data center $s$ during timeslot $t$ is

$$\mathcal{E}_{t,s} = f_{t,s}\left(x_{t,s}, \sum\nolimits_j \lambda_{t,j,s}\right). \tag{4.2}$$

**Switching cost**

For the switching cost, let $\beta_s$ be the cost to transition a server from the sleep state to the active state at data center $s$. The same as in Chapter 3, we assume that the cost of transitioning from the active to the sleep state is 0. If this is not the case, we can simply fold the corresponding cost into the cost $\beta_s$ incurred in the next power-up operation. Thus the switching cost for changing the number of active servers from $x_{t-1,s}$ to $x_{t,s}$ is

$$d(x_{t-1,s}, x_{t,s}) = \beta_s(x_{t,s} - x_{t-1,s})^+,$$

where $(x)^+ = \max(0, x)$. The constant $\beta_s$ includes the costs of (i) the energy used toggling a server, (ii) the delay in migrating state, such as data or a virtual machine (VM), when toggling a server, (iii) increased wear-and-tear on the servers toggling, and (iv) the risk associated with server toggling. If only (i) and (ii) matter, then $\beta_s$ is either on the order of the cost to run a server for a few seconds (waking from suspend-to-RAM or migrating network state [37] or storage state [108]), or several minutes (to migrate a large VM [39]). However, if (iii) is included, then $\beta_s$ becomes on the order of the cost to run a server for an hour [27]. Finally, if (iv) is considered then our conversations with operators suggest that their perceived risk that servers will not turn on properly when toggled is high, so $\beta_s$ may be even larger.

### 4.1.3 Cost optimization problem

Given the workload and cost models above, we model the Internet-scale system as a cost-minimizer. In particular, we formalize the goal of the Internet-scale system as choosing the routing policy $\lambda_{t,j,s}$ and the number of active servers $x_{t,s}$ at each time $t$ so as to minimize the total cost during $[1, T]$. This can be written as follows:

$$\begin{aligned} \underset{x_{t,s},\lambda_{t,j,s}}{\text{minimize}} \quad & \sum_{t=1}^{T}\sum_{s=1}^{S} \mathcal{E}_{t,s} + \mathcal{D}_{t,s} + d(x_{t-1,s}, x_{t,s}) \\ \text{subject to} \quad & \sum\nolimits_{s=1}^{S} \lambda_{t,j,s} = \lambda_{t,j}, \quad \forall t, \forall j \\ & \lambda_{t,j,s} \geq 0, \quad \forall t, \forall j, \forall s \\ & 0 = x_{0,s} \leq x_{t,s} \leq M_s, \quad \forall t, \forall s \end{aligned} \tag{4.3}$$

The above optimization problem is jointly convex in $\lambda_{t,j,s}$ and $x_{t,s}$, thus in many cases the solution can be found easily *offline*, i.e., given all the information in $[1, T]$. However, our goal is to find *online* algorithms for this optimization, i.e., algorithms that determine $\lambda_{t,j,s}$ and $x_{t,s}$ using only information up to time $t + w$ where $w \geq 0$ is called the "prediction window". Based on the structure of optimization (4.3), we can see that $\lambda_{t,j,s}$ can be solved easily at timeslot $t$ once $x_{t,s}$ are fixed. Thus the challenge for the online algorithms is to decide $x_{t,s}$ online.

### 4.1.4 Generalizations

Although the optimization problem (4.3) is very general already, the online algorithms and results in this chapter additionally apply to the following, more general framework:

$$\begin{aligned} \underset{x_1,\ldots,x_T}{\text{minimize}} \quad & \sum_{t=1}^{T} h_t(x_t) + \sum_{t=1}^{T} d(x_{t-1}, x_t) \\ \text{subject to} \quad & 0 \leq x_t \in \mathbb{R}^S, \qquad x_0 = 0. \end{aligned} \tag{4.4}$$

where $x_t$ has a vector value and $\{h_t(\cdot)\}$ are convex functions. Importantly, this formulation can easily include various SLA constraints on mean queueing delay or the queueing delay violation probability. In fact, a variety of additional bounds on $x_t$ can be incorporated implicitly into the functions $h_t(\cdot)$ by extended-value extension, i.e., defining $h_t(\cdot)$ to be $\infty$ outside its domain. Clearly the optimization problem (4.4) is a vector version of the optimization problem (3.1) in the previous chapter.

To see how the optimization problem (4.3) fits into this general framework, we just need to define $h_t(x_t)$ for feasible $x_t$ as the optimal value to the following optimization over $\lambda_{t,j,s}$ given $x_{t,s}$ fixed:

$$\begin{aligned} \underset{\lambda_{t,j,s}}{\text{minimize}} \quad & \sum\nolimits_{s=1}^{S} (\mathcal{E}_{t,s} + \mathcal{D}_{t,s}) \\ \text{subject to} \quad & \sum\nolimits_{s=1}^{S} \lambda_{t,j,s} = \lambda_{t,j}, \quad \forall j \\ & \lambda_{t,j,s} \geq 0, \quad \forall j, \forall s \end{aligned} \tag{4.5}$$

For infeasible $x_t$ ($x_{t,s} \notin [0, M_s]$ for some $s$) we define $h_t(x_t) = \infty$. We can see that the optimal

workload dispatching has been captured by the definition of $h_t(x_t)$. Note that other restrictions of workload dispatching may be incorporated by the definition of $h_t(x_t)$ similarly.

Intuitively, this general model seeks to minimize the sum of a sequence of convex functions when "smooth" solutions are preferred, i.e., it is a *smoothed online convex optimization* problem. This class of problems has many important applications, including more general capacity provisioning in geographically distributed data centers, video streaming [66] in which encoding quality varies but large changes in encoding quality are visually annoying to users, automatically switched optical networks (ASONs) in which there is a cost for re-establishing a lightpath [126], and power generation with dynamic demand, since the cheapest types of generators typically have very high switching costs [69].

In order to evaluate the performance of the online algorithms we discuss, we focus on the *competitive ratio* again. Actually the geographical load balancing problem (4.3) and the generalization (4.4) are instances of the class of problems known as "metrical task systems (MTSs)". MTSs have received considerable study in the algorithms literature, and it is known that if no further structure is placed on them, then the best deterministic algorithm for a MTS has competitive ratio proportional to the number of system states [29], which is infinity in our problem.

Note that the analytic results of Section 4.2 focus on the competitive ratio, assuming that the service has a finite duration, i.e., $T < \infty$, but allowing arbitrary sequences of convex functions $\{h_t(\cdot)\}$. Thus, the analytic results provide worst-case (robustness) guarantees. However, to provide realistic cost estimates, we also consider case studies using real-world traces for $\{h_t(\cdot)\}$ in Section 4.3.

## 4.2 Algorithms and analytical results

We can now study and design online algorithms for geographical load balancing. We start by analyzing the performance of the classic receding horizon control (RHC). This uncovers some drawbacks of RHC, and so in the second part of this section we propose new algorithms which address these. We defer the proofs to the appendix of this chapter.

### 4.2.1 Receding horizon control

RHC is classical control policy [79] that has been proposed for dynamic capacity provisioning in data centers [116, 78].

Informally, RHC works by, at time $\tau$, solving the cost optimization over the window $(\tau, \tau + w)$ given the starting state $x_{\tau-1}$. Formally, define $X^\tau(x_{\tau-1})$ as the vector in $(\mathbb{R}^S)^{w+1}$ indexed by

$t \in \{\tau, \ldots, \tau + w\}$, which is the solution to

$$\begin{array}{ll} \underset{x_\tau, \ldots, x_{\tau+w}}{\text{minimize}} & \sum_{t=\tau}^{\tau+w} h_t(x_t) + \sum_{t=\tau}^{\tau+w} d(x_{t-1}, x_t) \\ \text{subject to} & 0 \leq x_t \in \mathbb{R}^S. \end{array} \tag{4.6}$$

**Algorithm 4.1** (Receding horizon control: RHC)**.** *For all $t \leq 0$, set the number of active servers to $x_{RHC,t} = 0$. At each timeslot $\tau \geq 1$, set the number of active servers to*

$$x_{RHC,\tau} = X_\tau^\tau(x_{RHC,\tau-1}).$$

In studying the performance of RHC there is a clear divide between the following two cases:

1. *The homogeneous setting ($S = 1$):* This setting considers only one class of servers, and thus corresponds to a single data center with homogeneous servers. Under this setting, only the number of active servers is important, not which servers are active, i.e., $x_t$ is a scalar (In Chapter 3, we have seen some numerical results of RHC compared with LCP in this scenario.).

2. *The heterogeneous setting ($S \geq 2$):* This setting allows for different types of servers, and thus corresponds to a single data center with heterogeneous servers or to a collection of geographically diverse data centers. Under this setting, we need to decide the number of active servers of each type, i.e., $x_t$ is a vector.

To start, let us focus on the homogeneous setting (i.e., the case of dynamic resizing capacity within a homogeneous data center). In this case, RHC performs well: it has a small competitive ratio that depends on the minimal cost of an active server and the switching cost, and decays to one quickly as the prediction window grows. Specifically:

**Theorem 4.1.** *In the homogeneous setting ($S = 1$), RHC is $(1 + \frac{\beta}{(w+1)e_0})$-competitive.*

Theorem 4.1 is established by showing that RHC is not worse than another algorithm which can be proved to be $(1 + \frac{\beta}{(w+1)e_0})$-competitive. Given Theorem 4.1, it is natural to wonder if the competitive ratio is tight. The following result highlights that there exist settings where the performance of RHC is quite close to the bound in Theorem 4.1.

**Theorem 4.2.** *In the homogeneous setting ($S = 1$), RHC is not better than $(\frac{1}{w+2} + \frac{\beta}{(w+2)e_0})$-competitive.*

It is interesting to note that [77] shows that a prediction window of $w$ can improve the performance of a metrical task system by a factor of at most $2w$. If $\beta/e_0 \gg 1$ then RHC is approximately within a factor of 2 of this limit in the homogeneous case.

The two theorems above highlight that, with enough lookahead, RHC is guaranteed to perform quite well in the homogeneous setting. Unfortunately, the story is different in the heterogeneous setting, which is required to model the geographical load balancing.

**Theorem 4.3.** *In the heterogeneous setting ($S \geq 2$), given any $w \geq 0$, RHC is $\geq (1+\max_s(\beta_s/e_{0,s}))$-competitive.*

In particular, for any $w > 0$ the competitive ratio in the heterogeneous setting is at least as large as the competitive ratio in the homogeneous setting with no predictions ($w = 0$). Most surprisingly (and problematically), this highlights that RHC may not see any improvement in the competitive ratio as $w$ is allowed to grow.

The proof, given in Appendix 4.D involves constructing a workload such that servers at different data centers turn on and off in a cyclic fashion under RHC, whereas the optimal solution is to avoid such switching. Therefore, $\{h_t(\cdot)\}$ resulting in bad competitive ratio are not any weird functions but include practical cost functions for formulation (4.3). Note that the larger the prediction window $w$ is, the larger the number of data centers must be in order to achieve this worst case.

The results above highlight that, though RHC has been widely used, RHC may result in unexpected bad performance in some scenarios, i.e., it does not have "robust" performance guarantees. The reason that RHC may perform poorly in the heterogeneous setting is that it may change provisioning due to (wrongly) assuming that the switching cost would get paid off within the prediction window. For the geographical load balancing case, the electricity price based on the availability of renewable power (e.g., wind or solar) may change dramatically during a short time period. It is very hard for RHC to decide which data centers to increase/decrease capacity without knowing the entire future information, thus RHC may have to change its decisions and shift the capacity among data centers very frequently, which results in a big switching cost. Notice that this does not happen in the homogeneous setting where we do not need to decide which data center to use, and the new information obtained in the following timeslots would only make RHC correct its decision monotonically (increase but not decrease the provisioning by Lemma 4.3).

In the rest of this section we propose an algorithm with significantly better robustness guarantees than RHC.

### 4.2.2 Fixed horizon control

In this section, we present a new algorithm, averaging fixed horizon control (AFHC), which addresses the limitation of RHC identified above. Specifically, AFHC achieves a competitive ratio for the heterogeneous setting that matches that of RHC in the homogeneous setting.

Intuitively, AFHC works by combining $w + 1$ different bad algorithms, which each use a fixed horizon optimization, i.e., at time 1 algorithm 1 solves and implements the cost optimization for

$[1, 1+w]$, at time 2 algorithm 2 solves and implements the cost optimization for $[2, 2+w]$, etc.

More formally, first consider a family of algorithms parameterized by $k \in [1, w+1]$ that recompute their provisioning periodically. For all $k = 1, \ldots, w+1$, let $\Omega_k = \{i : i \equiv k \mod (w+1)\} \cap [-w, \infty)$; this is the set of integers congruent to $k$ modulo $w+1$, such that the lookahead window at each $\tau \in \Omega_k$ contains at least one $t \geq 1$.

**Algorithm 4.2** (Fixed horizon control, version $k$: FHC$^{(k)}$)**.** *For all $t \leq 0$, set the number of active servers to $x^{(k)}_{FHC,t} = 0$. At timeslot $\tau \in \Omega_k$, for all $t \in \{\tau, \ldots, \tau + w\}$, use (4.6) to set*

$$x^{(k)}_{FHC,t} = X^{\tau}_t \left( x^{(k)}_{FHC,\tau-1} \right). \tag{4.7}$$

For notational convenience, we often set $x^{(k)} \equiv x^{(k)}_{FHC}$. Note that for $k > 1$ the algorithm starts from $\tau = k - (w+1)$ rather than $\tau = k$ in order to calculate $x^{(k)}_{FHC,t}$ for $t < k$.

FHC can clearly have very poor performance. However, surprisingly, by averaging different versions of FHC we obtain an algorithm with better performance guarantees than RHC. More specifically, AFHC is defined as follows.

**Algorithm 4.3** (Averaging FHC: AFHC)**.** *At timeslot $\tau \in \Omega_k$, use FHC$^{(k)}$ to determine the provisioning $x^{(k)}_{\tau}, \ldots, x^{(k)}_{\tau+w}$, and then set $x_{AFHC,t} = \sum_{n=1}^{w+1} x^{(n)}_t / (w+1)$.*

Intuitively, AFHC seems worse than RHC because RHC uses the latest information to make the current decision and AFHC relies on FHC which makes decisions in advance, thus ignoring some possibly valuable information. This intuition is partially true, as shown in the following theorem, which states that RHC is not worse than AFHC for any workload in the homogeneous setting ($S = 1$).

**Theorem 4.4.** *In the homogeneous setting ($S = 1$), cost(RHC) $\leq$ cost(AFHC).*

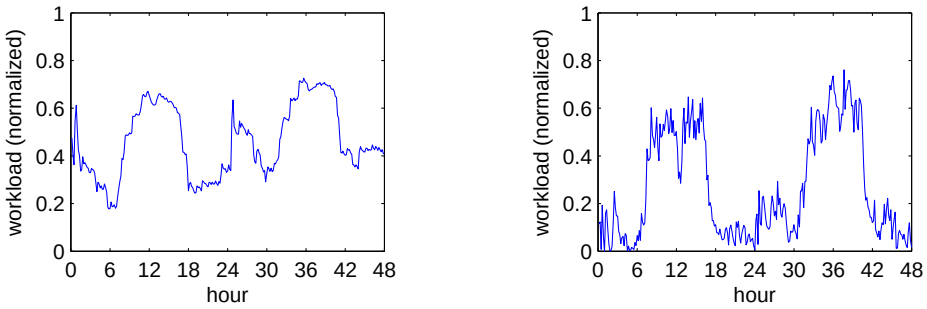
Though RHC is always better than AFHC in the homogeneous setting, the key is that AFHC can be significantly better than RHC in the heterogeneous case, even when $S = 2$.

**Theorem 4.5.** *In heterogeneous setting ($S \geq 2$), there exist convex functions $\{h_t(\cdot)\}$ such that*

$$cost(RHC) > cost(AFHC).$$

Moreover, the competitive ratio of AFHC is much better than that of RHC in the heterogeneous case.

**Theorem 4.6.** *AFHC is $\left(1 + \max_s \frac{\beta_s}{(w+1)e_{0,s}}\right)$-competitive in both the homogeneous setting and the heterogeneous setting.*




(a) Trace 1 (b) Trace 2

Figure 4.1: HP workload traces.

The contrast between Theorems 4.3 and 4.6 highlights the improvement AFHC provides over RHC. In fact, AFHC has the same competitive ratio in the general (possibly heterogeneous) case that RHC has in the homogeneous case. So, AFHC provides the same robustness guarantee for geographical load balancing that RHC can provide for a homogeneous local data center.

## 4.3 Case studies

In the remainder of the chapter, we provide a detailed study of the performance of the algorithms described in the previous section. Our goal is threefold: (i) to understand the performance of the algorithms (RHC and AFHC) in realistic settings; (ii) to understand the potential environmental benefits of using geographical load balancing to implement "follow the renewables" routing; and (iii) to understand the optimal portfolio of renewable sources for use within an Internet-scale system.

### 4.3.1 Experimental setup

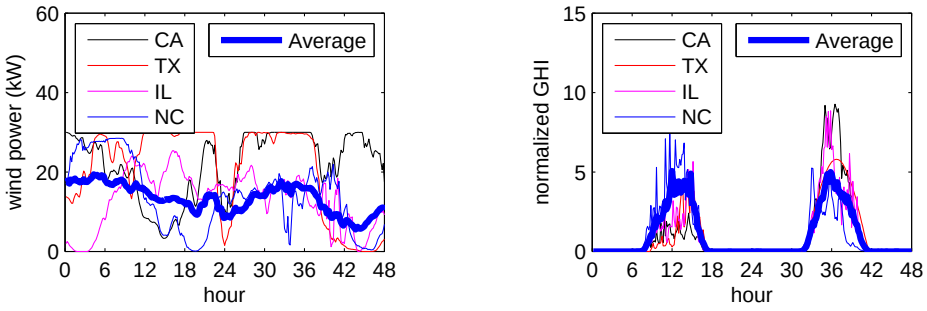
This study uses the setup similar to that of [88], based on real-world traces for data center locations, traffic workloads, renewable availability, energy prices, etc., as described below.[2]

#### The workload

We consider 48 sources of requests, with one source at the center of each of the 48 continental US states. We consider 10-minute time slots over two days.

The workload $\lambda_t$ is generated from two traces at Hewlett-Packard Labs [49] shown in Figure 4.1. These are scaled proportional to the number of Internet users in each state, and shifted in time to account for the time zone of that state.

[2]Note that the setup considered here is significantly more general than that of [88], as follows. Most importantly, [88] did not model switching costs (and so did not consider online algorithms). Additionally, the current work investigates the optimal renewable portfolio more carefully, using multiple traces and varying the renewable capacity among other things.




(a) Wind (b) Solar

Figure 4.2: Renewable generation for two days.

**The availability of renewable energy**

To capture the availability of solar and wind energy, we use traces with 10 minute granularity from [111, 112] for global horizontal irradiance (GHI) scaled to average 1, and power output of a 30 kW wind turbine. The traces of four states (CA, TX, IL, NC) are illustrated in Figure 4.2. Note that *we do not consider solar thermal*, because of the significant infrastructure it requires. Since these plants often incorporate a day's thermal storage [94], the results could be very different if solar thermal were considered.

These figures illustrate two important features of renewable energy: spatial variation and temporal variation. In particular, wind energy does not exhibit a clear pattern throughout the day and there is little correlation across the locations considered. In contrast, solar energy has a predictable peak during the day and is highly correlated across the locations.

In our investigation, we scale the "capacity" of wind and solar. When doing so, we scale the availability of wind and solar linearly, which models scaling the number of generators in a wind farm or solar installation, rather than the capacity of each. We measure the "capacity" $c$ of renewables as the ratio of the *average* renewable generation to the minimal energy required to serve the average workload. Thus, $c = 2$ means that the average renewable generation is twice the minimal energy required to serve the average workload. We set capacity $c = 1$ by default, but vary it in Figures 4.5 and 4.7.

**The Internet-scale system**

We consider the Internet-scale system as a set of 10 data centers, placed at the centers of states known to have Google data centers [92], namely California, Washington, Oregon, Illinois, Georgia, Virginia, Texas, Florida, North Carolina, and South Carolina. Data center $s$ contains $M_s$ homogeneous servers, where $M_s$ is set to be twice the minimal number of servers required to serve the peak workload of data center $s$ under a scheme which routes traffic to the nearest data center. Further,

the renewable availability at each data center is defined by the wind/solar trace from a nearby location, usually within the same state.

We set the energy cost as the number of active servers *excluding* those that can be powered by renewables. Note that this assumes that data centers operate their own wind and solar generations and pay no marginal cost for renewable energy. Further, it ignores the installation and maintenance costs of renewable generation. Quantitatively, if the renewable energy available at data center $s$ at time $t$ is $r_{t,s}$, measured in terms of number of servers that can be powered, then the energy cost of data center $s$ at time $t$ is

$$\mathcal{E}_{t,s} = p_s(x_{t,s} - r_{t,s})^+. \tag{4.8}$$

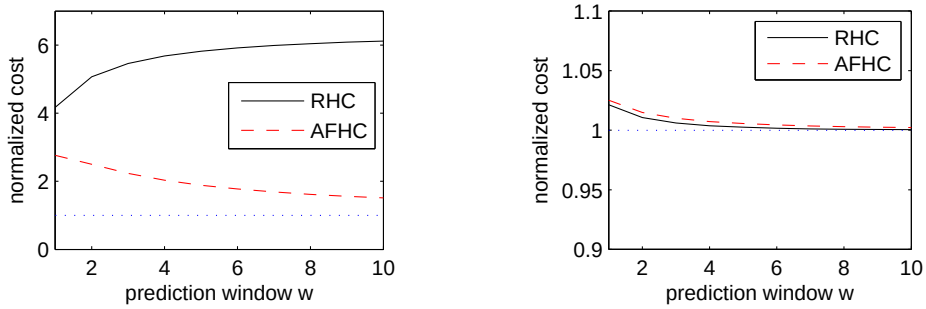
Here $p_s$ for each data center is constant, and equals to the industrial electricity price of each state in May 2010 [113]. This contrasts with the total power cost $p_s x_{t,s}$ typically used without owning renewable generation.

For delay cost, we set the round-trip network delay $\delta_{t,j,s}$ to be proportional to the distance between source and data center plus a constant (10 ms), resulting in round-trip delays in [10 ms, 260 ms]. We model the queueing delays using parallel M/GI/1/processor sharing queues with the total load $\sum_j \lambda_{t,j,s}$ divided equally among the $x_{t,s}$ active servers, each having service rate $\mu_s = 0.2(\text{ms})^{-1}$. Therefore, the delay cost of data center $s$ at time $t$ is

$$\mathcal{D}_{t,s} = \gamma \sum_j \lambda_{t,j,s} \left( \frac{1}{\mu_s - \sum_j \lambda_{t,j,s}/x_{t,s}} + \delta_{t,j,s} \right). \tag{4.9}$$

Here we consider linear lost revenue function $r_t(d) = \gamma d$, where $\gamma$ is set to be 1. Measurements [56] show that a 500 ms increase in delay reduces revenue by 20%, or 0.04%/ms. To get a conservative estimate of the benefit from geographical load balancing, we pick $\gamma = 1$, which is slightly higher than [56], so that the penalty for the propagation delay of geographical load balancing is high compared to the benchmark policy. Later we scale $p_s$ (with $\gamma = 1$ corresponding to the default setting) in Figure 4.4(a) to show the impact of relative energy cost to delay cost as energy price possibly goes high in future, or the delay penalty is lower for the systems.

For the switching cost, we set $\beta = 6$ by default, which corresponds to the operating cost of an idle server for about half an hour to one hour. We vary $\beta$ in Figure 4.4(b) to show its impact on cost saving. For the prediction window, we set $w = 3$ by default, which corresponds to half an hour prediction of workload and renewable generation. We vary $w$ in Figure 4.3 to examine its impact on cost saving.




(a) Cost using workload used in the proof of Theorem 4.3

(b) Cost using Trace 1

Figure 4.3: Total cost, normalized by the cost of OPT, versus prediction window under RHC and AFHC

#### Algorithms

We use optimization (4.3) with energy cost (4.8) and delay cost (4.9) for the geographical load balancing. We use "GLB" to denote the offline optimal solution to (4.3). The online solutions of algorithms receding horizon control and averaging fixed horizon control are denoted by "RHC" and "AFHC", respectively.

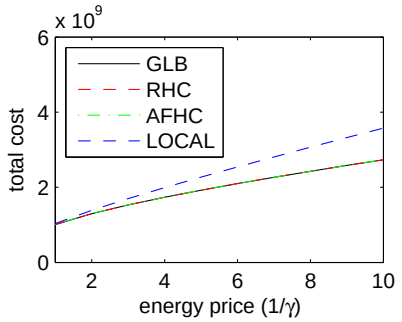
As a benchmark for comparison, we consider a system that does no geographical load balancing, but instead routes requests to the nearest data center and optimally adjusts the number of active servers at each location. We call this system 'LOCAL' and use it to illustrate the benefits that come from using geographical load balancing.

### 4.3.2 Experimental results

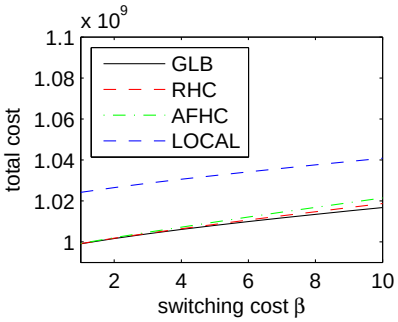
With the foregoing setup, we performed several numerical experiments to evaluate the feasibility of moving toward Internet-scale systems powered (nearly) entirely by renewable energy and the optimal portfolios.

#### The performance of RHC and AFHC

Geographical load balancing is known to provide Internet-scale system operators cost savings. Let us first study the cost saving from geographical load balancing and how much of it can be achieved by the online algorithms RHC and AFHC. Figure 4.3(a) shows the total cost in the bad scenario with an artificial workload used in the proof of Theorem 4.3 (with $J = (w+1)^2$ types of jobs), which illustrates that AFHC may have much better performance in the worst case. The degradation in the performance of RHC as $w$ grows is because $J$ also grows. In contrast, Figure 4.3(b) shows the total cost of RHC and AFHC (with default settings but $\beta = 6\min(p_s)$, the same as in the bad scenario) under HP Trace 1. We can see that both RHC and AFHC are nearly optimal for the real

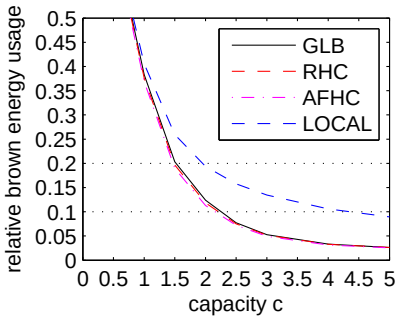



(a) Effect of energy price, with switching cost $\beta = 6$.

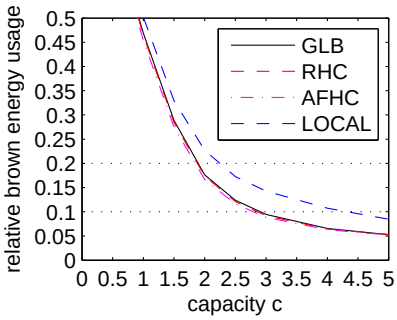



(b) Effect of switching cost with delay cost $\gamma = 1$.

Figure 4.4: Impact of the energy price and switching cost when the total renewable capacity is $c = 1$.




(a) Trace 1




(b) Trace 2

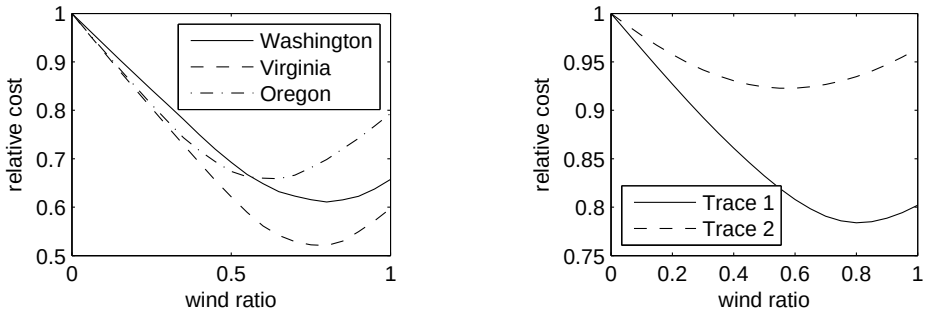
Figure 4.5: Impact of the renewable capacity when solar percentage is 20%.

workload. Figure 4.3 confirms that AFHC is able to provide worst-case guarantee without giving up much performance in common cases.

This behavior under real workload is further illustrated in Figure 4.4, which shows the total cost under GLB, RHC, AFHC, and LOCAL as energy price or switching cost is increased. The cost saving of GLB over LOCAL becomes large when the energy price is high because GLB can save a great deal of energy cost at the expense of small increases in network delay since requests can be routed to where energy is cheap or renewable generation is high. Moreover, the cost saving of GLB, RHC and AFHC over LOCAL looks stable for a wide range of switching cost.

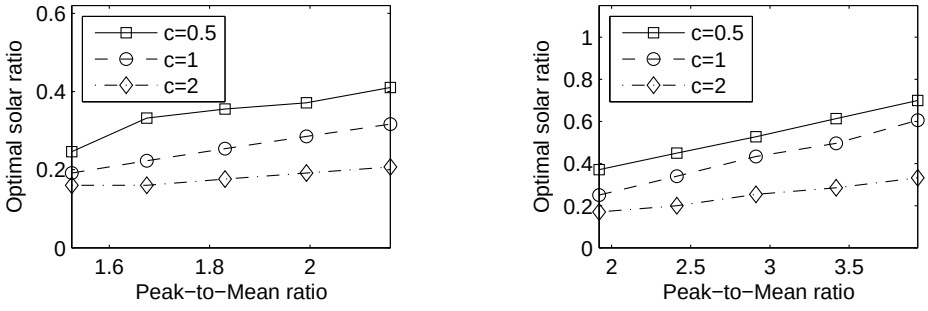
### The impact of geographical load balancing

Geographical load balancing is much more efficient at using renewable supply than LOCAL because it can route traffic to the data center with higher renewable generation. Figure 4.5 illustrates the differences of brown energy usage as a function of the capacity of renewable energy for both traces. The brown energy consumption is scaled so that the consumption is 1 when there is no renewable




(a) LOCAL using Trace 1 (b) GLB

Figure 4.6: Impact of the mix of renewable energy used.




(a) Trace 1 (b) Trace 2

Figure 4.7: Optimal portfolios for different PMRs and capacities.

($c = 0$). Interestingly, Figure 4.5 highlights that when there is little capacity of renewables, both GLB and LOCAL can take advantage of it, but that as the capacity of renewables increases GLB is much more efficient at using it, especially for Trace 1. This is evident by the significantly lower brown energy consumption of GLB that emerges at capacities $> 1$. For Trace 1 in Figure 4.5(a), the capacities of renewables necessary to reduce brown energy usage to 20% and 10% under LOCAL are 1.9 and 4.3, respectively, while those required under GLB are only 1.5 and 2.3. Similar reductions can be observed for Trace 2 in Figure 4.5(b).

As in Figures 4.3(b) and 4.4, the performance of RHC and AFHC is again quite close to that of the optimal solution GLB, which reinforces that both RHC and AFHC are nearly optimal in common cases. Therefore we will show only GLB and LOCAL for the remaining experiments.

### The optimal renewable portfolio

We now move to the question of what mix of solar and wind is most effective. A priori, it seems that solar may be the most effective, since the peak of solar availability is closely aligned with that of the data center workload. However, the fact that solar is not available during the night is a significant

drawback, which makes wind necessary to power the data centers during night. Our results lend support to the discussion above. For each data center under LOCAL, the optimal wind percentage is quite different for each location because of different renewable generation qualities, as shown in Figure 4.6(a). There are also similarities for different locations, e.g., the optimal portfolios contain both solar and wind, and wind has a large percentage, 60%- 90%.

Once GLB is used, it becomes possible to aggregate wind availability across geographical locations. This makes wind more valuable since wind is not correlated across large geographical distances, and so when aggregated, the availability smoothes. As illustrated in Figure 4.6(b), the optimal renewable portfolio for Trace 1 contains 80% wind. We can also see that the optimal portfolio is affected significantly by the workload characteristics. Compared with Trace 1, Trace 2 has less base load during night, requiring less wind generation.

The impact of workload characteristics becomes more clear in Figure 4.7, where we use loads $\lambda'_{j,t} = \lambda^{\alpha}_{j,t}, \alpha = 1, 1.25, 1.5, 1.75, 2$, to get different peak-to-mean ratios. For large diurnal peak-to-mean ratios the optimal portfolio can be expected to use a higher percentage of solar because solar peak is closely aligned with the workload peak, which is validated in Figure 4.7. Also, when renewable capacity is fairly large and we plan to install extra capacity, since solar generation can already provide enough power to serve the workload peak around noon, the increased renewable capacity can then be almost from wind generation to serve the workload during other times, especially night. This will make the solar ratio lower in the optimal portfolios, which can be seen from the lines of different renewable capacities in Figure 4.7.

## 4.4 Concluding remarks

This chapter studies online algorithms for geographical load balancing problem in Internet-scale systems via both theoretical analysis and trace-based experiments. The optimization problem in this chapter can be seen as a vector generalization of the optimization problem in the previous chapter. We show that the classical algorithm, receding horizon control (RHC), works well in homogeneous setting (where all servers are equally able to serve every request). However, in general, RHC can perform badly for the heterogeneous settings needed for geographical load balancing. Motivated by the weakness of RHC, we design a new algorithm called averaging fixed horizon control (AFHC) which guarantees good performance. We evaluate RHC and AFHC under workloads measured on a real data center. The numerical results show that RHC and AFHC are nearly optimal for our traces, which implies that the improvement in worst-case performance of AFHC comes at negligible cost to the average-case performance. The experiments also reveal vital role of geographical load balancing in reducing brown energy consumption to (nearly) zero. We also perform a detailed study on the impact of workload characteristics and renewable capacity on the optimal renewable portfolio under

GLB.

## Appendix 4.A Notation

We first introduce some additional notation used in the proofs. For brevity, for any vector $y$ we write $y_{i..j} = (y_i, \dots, y_j)$ for any $i \leq j$.

Let $x^*$ denote the offline optimal solution to optimization (4.4), and OPT be the algorithm that uses $x^*$. Further, let $X$ be the result of RHC, and recall that $x^{(k)}$ is the result of FHC$^{(k)}$.

Let the cost during $[t_1, t_2]$ with boundary conditions be

$$g_{t_1,t_2}(x; x_S; x_E) = \sum_{t=t_1}^{t_2} h_t(x_t) + d(x_S, x_{t_1}) + \sum_{t=t_1+1}^{t_2} d(x_{t-1}, x_t) + d(x_{t_2}, x_E).$$

If $x_E$ is omitted, then by convention $x_E = 0$ (and thus $d(x_{t_2}, x_E) = 0$). If $x_S$ is omitted, then by convention $x_S = x_{t_1-1}$. Note that $g_{t_1,t_2}(x)$ depends on $x_i$ only for $t_1 - 1 \leq i \leq t_2$.

For any algorithm $A \in \{RHC, FHC, AFHC, OPT\}$, the total cost is $cost(A) = g_{1,T}(x_A)$.

## Appendix 4.B Proof of Theorems 4.6

**Lemma 4.1.** *Since $d(\cdot)$ satisfies triangle inequality, we have*

$$cost(FHC^{(k)}) \leq cost(OPT) + \sum_{\tau \in \Omega_k} d(x^{(k)}_{\tau-1}, x^*_{\tau-1}).$$

*Proof.* For every $k = 1, \dots, w+1$ and every $\tau \in \Omega_k$,

$$\begin{aligned} g_{\tau,\tau+w}(x^{(k)}) =& \sum_{t=\tau}^{\tau+w} h_t(x^{(k)}_t) + \sum_{t=\tau}^{\tau+w} d(x^{(k)}_{t-1}, x^{(k)}_t) \\ \leq& \sum_{t=\tau}^{\tau+w} h_t(x^*_t) + \sum_{t=\tau+1}^{\tau+w} d(x^*_{t-1}, x^*_t) + d(x^{(k)}_{\tau-1}, x^*_{\tau-1}) + d(x^*_{\tau-1}, x^*_\tau) \\ =& g_{\tau,\tau+w}(x^*) + d(x^{(k)}_{\tau-1}, x^*_{\tau-1}). \end{aligned} \tag{4.10}$$

Summing the above over $\tau \in \Omega_k$, establishes the lemma. □

*Proof of Theorem 4.6.* Substituting $d(x, y) = \beta \cdot (y - x)^+$ into Lemma 4.1, by the convexity of $h_t$

(and thus $g_{1,T}$),

$$\begin{aligned}\frac{cost(AFHC)}{cost(OPT)} &\le \frac{1}{w+1}\sum_{k=1}^{w+1}\frac{g_{1,T}(x^{(k)})}{cost(OPT)} \le 1+\frac{\beta\cdot\sum_{t=1}^{T}x^*_{t-1}}{(w+1)cost(OPT)}\\ &\le 1+\frac{\beta\cdot\sum_{t=1}^{T}x^*_{t-1}}{(w+1)\sum_{t=1}^{T}h_t(x^*_t)} \le 1+\frac{\beta\cdot\sum_{t=1}^{T}x^*_{t-1}}{(w+1)e_0\cdot\sum_{t=1}^{T}x^*_t}\\ &\le 1+\max_s\frac{\beta_s}{(w+1)e_{0,s}}\end{aligned}$$

where the second step uses Lemma 4.1, and the last step uses the facts that $\beta_i/e_{0,i} \le \max_s \beta_s/e_{0,s}$, and $0 \le \sum_{t=1}^{T} x^*_{t-1} \le \sum_{t=1}^{T} x^*_t$ element-wise as $x^*_0 = 0$. □

## Appendix 4.C Proofs of Theorems 4.1 and 4.4

The following lemma says that the optimal solution on $[i, j]$ is non-decreasing in the initial condition $x_{i-1}$ and the final condition $x_{j+1}$.

**Lemma 4.2.** *Let $S = 1$. Given constants $x_{i-1}, x_{j+1} \in \mathbb{R}$, let $x^{ij} = (x^{ij}_i, \dots, x^{ij}_j)$ be a vector minimizing $g_{i,j}(x; x_{i-1}; x_{j+1})$. Then for any $\hat{x}_{i-1} \ge x_{i-1}$ and $\hat{x}_{j+1} \ge x_{j+1}$, there exists a vector $\hat{x}^{ij} = (\hat{x}^{ij}_i, \dots, \hat{x}^{ij}_j)$ minimizing $g_{i,j}(x; \hat{x}_{i-1}; \hat{x}_{j+1})$ such that $\hat{x}^{ij} \ge x^{ij}$.*

*Proof.* Since $x^{ij}$ and $\hat{x}^{ij}$ minimize their respective objectives, we have $g_{i,j}(x^{ij}; x_{i-1}; x_{j+1}) \le g_{i,j}(\hat{x}^{ij}; x_{i-1}; x_{j+1})$ and $g_{i,j}(\hat{x}^{ij}; \hat{x}_{i-1}; \hat{x}_{j+1}) \le g_{i,j}(x^{ij}; \hat{x}_{i-1}; \hat{x}_{j+1})$. If there is an $x^{ij}$ such that the latter holds with equality, then we can choose $\hat{x}^{ij}_i = x^{ij}_i$ and consider the problem with $g_{i+1,j}$ recursively. Otherwise, i.e., the latter is a strict inequality, summing the two inequalities and canceling terms gives

$$\begin{aligned}&(x^{ij}_i - x_{i-1})^+ + (\hat{x}^{ij}_i - \hat{x}_{i-1})^+ + (\hat{x}_{j+1} - \hat{x}^{ij}_j)^+ + (x_{j+1} - x^{ij}_j)^+\\ <&(\hat{x}^{ij}_i - x_{i-1})^+ + (x^{ij}_i - \hat{x}_{i-1})^+ + (\hat{x}_{j+1} - x^{ij}_j)^+ + (x_{j+1} - \hat{x}^{ij}_j)^+.\end{aligned}$$

Since $\hat{x}_{i-1} \ge x_{i-1}$ and $\hat{x}_{j+1} \ge x_{j+1}$, it follows that either $x^{ij}_i < \hat{x}^{ij}_i$ or $x^{ij}_j < \hat{x}^{ij}_j$, by the submodularity of $\phi(x, y) = (x - y)^+$. In either case, we can continue recursively, considering $g_{i+1,j}$ in the former case or $g_{i,j-1}$ in the latter.

Finally we have $\hat{x}^{ij} \ge x^{ij}$. □

The next technical lemma says that RHC has larger solutions than related algorithms that look less far ahead.

**Lemma 4.3.** *Consider a system in the homogeneous setting ($S = 1$), and constants $t$, $X_{t-1} \ge \tilde{x}_{t-1} \ge 0$, and $k \in [t, t + w]$. Let $\tilde{x} = (\tilde{x}_t, \dots, \tilde{x}_k)$ minimize $g_{t,k}(x; \tilde{x}_{t-1})$, and let $X = x_{RHC}$. Then $\tilde{x} \le X_{t..k}$.*

*Proof.* The proof is by induction. By hypothesis, $\tilde{x}_{t-1} \leq X_{t-1}$. We need to prove that if $\tilde{x}_{\tau-1} \leq X_{\tau-1}$, then $\tilde{x}_\tau \leq X_\tau$.

Notice that $\tilde{x}_\tau$ is the first entry of a vector minimizing $g_{\tau,k}(x; x_{\tau-1})$. Similarly $X_\tau$ is the first entry of a vector minimizing $g_{\tau;\tau+w}(x, X_{\tau-1})$. If $k = \tau + w$, we have $\tilde{x}_\tau \leq X_\tau$ by Lemma 4.2 and the tie-break rule of RHC. Otherwise, i.e., $k < \tau + w$, we know that $X_\tau$ is the first entry of a vector minimizing $g_{\tau,k}(x; X_{\tau-1}; x'_{k+1})$ with $x'_{k+1} \geq 0$. By Lemma 4.2 and the RHC tie-break we again have $\tilde{x}_\tau \leq X_\tau$. □

Next comes the first main lemma used to prove Theorem 4.4.

**Lemma 4.4.** *In the homogeneous setting ($S = 1$), each version $k$ of FHC allocates fewer servers than RHC:*

$$x^{(k)}_{FHC} \leq x_{RHC}. \tag{4.11}$$

*Hence $x_{AFHC} \leq x_{RHC}$.*

*Proof.* Let $X = x_{RHC}$ be the result of RHC, and $x = x^{(k)}_{FHC}$. The proof is by induction. By definition, $x_0 = X_0 = 0$. To see that $x_{\tau-1} \leq X_{\tau-1}$ implies $x_\tau \leq X_\tau$, notice that $x_\tau$ is the first entry of a vector minimizing $g_{\tau,k}(x; x_{\tau-1})$ for some $k \in [\tau, \tau+w]$, with $k+1 \in \Omega_k$. The implication follows by Lemma 4.3 and establishes (4.11). The proof for $x_{AFHC}$ is immediate. □

**Lemma 4.5.** *In the homogeneous setting ($S = 1$), for any given vector $x \leq x_{RHC}$, we have $g_{1,T}(x_{RHC}) \leq g_{1,T}(x)$.*

*Proof.* Denote $X = x_{RHC}$. It is sufficient to construct a sequence of vectors $\xi^\tau$ such that: $\xi^1 = x$, $\xi^\tau_t = X_t$ for $t < \tau$, and $g_{1,T}(\xi^\tau)$ is non-increasing in $\tau$. The sequence can be constructed inductively with the additional invariant $\xi^\tau \leq X$ as follows.

At stage $\tau$, we calculate $\xi^{\tau+1}$. Apply RHC to get $X^\tau(X_{\tau-1}) = (\tilde{x}_\tau, \ldots, \tilde{x}_{\tau+w})$. Note that $\tilde{x}_\tau = X_\tau \geq \xi^\tau_\tau$ since $\xi^\tau \leq X$ by the inductive hypothesis. Moreover $\tilde{x}_{\tau..\tau+w} \leq X_{\tau..\tau+w}$ by Lemma 4.3.

If $\tilde{x}_{\tau..\tau+w} \geq \xi^\tau_{\tau..\tau+w}$ element-wise, then replace elements $\tau$ to $\tau + w$ in $\xi^\tau$ to get $\xi^{\tau+1} = (\xi^\tau_{1..\tau-1}, \tilde{x}_{\tau..\tau+w}, \xi^\tau_{\tau+w+1..T}) \geq \xi^\tau$. Then $g_{1,\tau+w}(\xi^{\tau+1}) \leq g_{1,\tau+w}(\xi^\tau)$ by the optimality of $\tilde{x}_{\tau..\tau+w}$. Since $\xi^\tau_{\tau+w} \leq \tilde{x}_{\tau+w}$ and $d(x, y) = \beta(y - x)^+$ is non-increasing in its first argument, we also have $g_{\tau+w+1,T}(\xi^{\tau+1}) \leq g_{\tau+w+1,T}(\xi^\tau)$. Therefore, $g_{1,T}(\xi^{\tau+1}) \leq g_{1,T}(\xi^\tau)$. Finally, to see that $\xi^{\tau+1} \leq X$, note that $\xi^\tau \leq X$ and $\tilde{x}_{\tau..\tau+w} \leq X_{\tau..\tau+w}$ as remarked above.

Otherwise, let $k \in [\tau+1, \tau+w]$ be the minimum index that $\tilde{x}_k < \xi^\tau_k$. Let $\xi^{\tau+1} = (\xi^\tau_{1..\tau-1}, \tilde{x}_{\tau..k-1}, \xi^\tau_{k..T}) \geq \xi^\tau$. Note that $k \geq \tau + 1$ since $\tilde{x}_\tau = X_\tau$; this ensures $\xi^\tau_t = X_t$ for $t < \tau$. Again, $\xi^{\tau+1} \leq X$ as in the previous case. It remains to prove $g_{1,T}(\xi^{\tau+1}) \leq g_{1,T}(\xi^\tau)$.

Let $u_A = \xi^\tau_{\tau..k-1}$, $u_B = \xi^\tau_{k..\tau+w}$, $\tilde{u}_A = \tilde{x}_{\tau..k-1}$ and $\tilde{u}_B = \tilde{x}_{k..\tau+w}$. Let vectors $(u_A, u_B)$, $(\tilde{u}_A, u_B)$, $(u_A, \tilde{u}_B)$ and $(\tilde{u}_A, \tilde{u}_B)$ be indexed by $t \in \{\tau, \ldots, \tau + w\}$. To see how replacing $\xi^\tau_{\tau\ldots k-1}$ by $\tilde{x}_{\tau\ldots k-1}$

affects the cost in $[1, T]$, note

$$g_{1,T}(\xi^{\tau+1}) - g_{1,T}(\xi^{\tau}) = g_{\tau,\tau+w}((\tilde{u}_A, u_B)) - g_{\tau,\tau+w}((u_A, u_B)).$$

Now since $\tilde{x}_k < \xi_k^\tau$, $\tilde{x}_{k-1} \geq \xi_{k-1}^\tau$ and $\phi(x, y) = (x - y)^+$ is submodular, we have

$$\begin{aligned} &(g_{\tau,\tau+w}((\tilde{u}_A, u_B)) - g_{\tau,\tau+w}((u_A, u_B))) + (g_{\tau,\tau+w}((u_A, \tilde{u}_B)) - g_{\tau,\tau+w}((\tilde{u}_A, \tilde{u}_B))) \qquad (4.12)\\ &= \beta\big((\xi_k^\tau - \tilde{x}_{k-1})^+ - (\xi_k^\tau - \xi_{k-1}^\tau)^+ + (\tilde{x}_k - \xi_{k-1}^\tau)^+ - (\tilde{x}_k - \tilde{x}_{k-1})^+\big)\\ &\leq 0. \end{aligned}$$

But since $(\tilde{u}_A, \tilde{u}_B)$ optimizes (4.6), we have

$$g_{\tau,\tau+w}((u_A, \tilde{u}_B)) - g_{\tau,\tau+w}((\tilde{u}_A, \tilde{u}_B)) \geq 0.$$

Thus the first bracketed term in (4.12) is non-positive, whence

$$g_{1,T}(\xi^{\tau+1}) - g_{1,T}(\xi^{\tau})) \leq g_{\tau,\tau+w}((\tilde{u}_A, u_B)) - g_{\tau,\tau+w}((u_A, u_B)) \leq 0.$$

□

*Proof of Theorem 4.4.* By Lemma 4.4 and AFHC, we have $\hat{x} \leq x_{RHC}$. By Lemma 4.5, $g_{1,T}(x_{RHC}) \leq g_{1,T}(\hat{x})$. □

*Proof of Theorem 4.1.* The bound on the competitive ratio of RHC follows from Theorems 4.4 and 4.6. □

## Appendix 4.D "Bad" instances for receding horizon control (RHC)

We now prove the lower bound results in Section 4.2 by constructing instances that force RHC to incur large costs.

*Proof of Theorem 4.2.* Consider the operating cost $h_t(x_t) = e_0 x_t$ for $\lambda_t \leq x_t$ and $h_t(x_t) = \infty$ for $\lambda_t > x_t$. Note that this cost function is convex. Now consider the arrival pattern $\lambda = \{\lambda_t\}_{1 \leq t \leq T}$ where $\lambda_{k(w+2)+1} = \Lambda > 0$ for $k = 0, 1, \ldots$ and other $\lambda_t$ are all 0. It is easy to see that $RHC$ will give the provisioning $X_{k(w+2)+1} = \Lambda$ and $X_t = 0$ for other $t$. Thus we have

$$g_{1,T}(X) = \frac{T}{w+2}\Lambda e_0 + \frac{T}{w+2}\beta\Lambda.$$

Now consider another provisioning policy $\hat{x} = \{\hat{x}_t = \Lambda\}_{1 \le t \le T}$. Its cost is $g_{1,T}(\hat{x}) = T\Lambda e_0 + \beta\Lambda$. Thus

$$g_{1,T}(X)/g_{1,T}(x^*) \ge g_{1,T}(X)/g_{1,T}(\hat{x}) = \frac{e_0 + \beta}{(w+2)(e_0 + \beta/T)} \sim \frac{1}{w+2} + \frac{\beta}{(w+2)e_0}$$

as $T \to \infty$. □

Note that the cost function in the proof of Theorem 4.2 is applicable to data centers that impose a maximum load on each server (to meet QoS or SLA requirements).

*Proof of Theorem 4.5.* When $S = 2$, the following geographical load balancing instance causes $cost(AFHC) < cost(RHC)$.

Choose constants $f_1 > f_2$ and $\beta_1 < \beta_2$ such that $(w+1)f_1 < (w+1)f_2 + \beta_2 < (w+1)f_1 + \beta_1$ and $wf_1 + \beta_1 < wf_2 + \beta_2$. These can simultaneously be achieved by choosing an arbitrary $f_1 - f_2 > 0$, then choosing $\beta_2 - \beta_1 \in (w, w+1)(f_1 - f_2)$, and then $\beta_2 > (w+1)(f_1 - f_2)$.

Let the switching cost for data center $i$ be $\beta_i$. Let the operating cost be $h_t(x_t) = f_1 x_{t,1} + f_2 x_{t,2}$ for $\lambda_t \le x_{t,1} + x_{t,2}$ and $h_t(x_t) = \infty$ for $\lambda_t > x_{t,1} + x_{t,2}$. Note that this function is convex. In this system, the servers in the second data center have lower operating cost but higher switching cost (e.g., more expensive, energy-efficient severs).

Choose constants $T > w + \max(1, \beta_2/(f_1 - f_2))$, and $\Lambda > 0$. Now consider the cost of schemes AFHC and RHC under the load such that: (a) $\lambda_{w+1} = 0$, (b) $\lambda_t = \Lambda$ for all other $t \in [1, T]$, and (c) $\lambda_t = 0$ for $t \notin [1, T]$.

**Under AFHC:** At time $t = 1$, $FHC^{(1)}$ sees $\lambda_1, \ldots, \lambda_{w+1}$ and so uses $\Lambda$ servers in data center 1 for the first $w$ timeslots and turns off all servers at timeslot $w+1$. From timeslot $t = w+2$ onwards, it sees $\lambda = \Lambda$ until time $T$, and so uses $\Lambda$ servers in data center 2 until $T$.

For $2 \le i \le w+1$, $FHC^{(i)}$ initially sees a window of loads in which $w$ or fewer time slots have non-zero load, and so again chooses servers in data center 1. However, the last slot in the first window, slot $i-1$, has load $\Lambda$, and so servers remain on. In the second and subsequent windows, the cost of switching is greater than uses servers in data center 1 until $T$. Thus its total cost is

$$cost(AFHC) = \frac{w}{w+1}(f_1\Lambda T + \beta_1\Lambda) + \frac{1}{w+1}(f_1\Lambda w + \beta_1\Lambda + f_2\Lambda(T - w - 1) + \beta_2\Lambda).$$

**Under RHC:** RHC uses only servers in data center 1 forever, for the same reason as $FHC^{(i)}$ for $2 \le i \le w+1$. Thus its total cost is

$$cost(RHC) = f_1\Lambda T + \beta_1\Lambda$$

The choice of $T$ implies $f_1(T - w) > f_2(T - w - 1) + \beta_2$, and thus $cost(AFHC) < cost(RHC)$. □

*Proof of Theorem 4.3.* The proof will be by construction. Consider an Internet-scale system with $S$ data centers and $J$ types of jobs (e.g., workload from different locations). Let $J \geq S \gg w$. Let the switching cost for servers in data center $s$ be $\beta_s = \beta_0 + 2\epsilon sw$. Denote the type-$j$ workload at time $t$ by $\lambda_{t,j}$ ($j \in \{1, \ldots, J\}$). Let the operating cost be $h_t(x_t) = \sum_{s=1}^{S}(e_0 - s\epsilon + C\sum_{l=s+1}^{J}\lambda_{t,l})x_{t,s}$ for $\sum_{s=1}^{S} x_{t,s} \geq \sum_{j=1}^{J}\lambda_{t,j}$ and $h_t(x_t) = \infty$ otherwise, where $\epsilon > 0$ is a small constant and $C > \max_s \beta_s$ is a large constant. Intuitively, this operating cost function means that servers in data center $s$ consume a little bit more energy when $s$ is smaller, and they are very inefficient at processing workload of types higher than $s$. Also, the switching cost increases slightly as $s$ increases. This may occur if all servers use roughly the same hardware, but data center $s$ store locally only data for jobs of types 1 to $s$.

Consider the workload trace which has $\lambda_{t,1} = \Lambda$ for $t = 1, \ldots, w+1$ and $\lambda_{t,t-w} = \Lambda$ for $t = w+2, \ldots, w+S$. All the other arrival rates $\lambda_{t,j}$ are zero. Then RHC would start with $\Lambda$ servers in data center 1 (the cheapest to turn on) at timeslot 1, and then at each $t \in [2, S]$ would switch off servers in data center $(t-1)$ and turn on $\Lambda$ servers in data center $t$ (the cheapest way to avoid the excessive cost of processing type $t$ jobs using servers in data center $s$ with $s < t$). For sufficiently small $\epsilon$, the optimal solution always uses $\Lambda$ servers in data center $S$ for $t \in [1, w+S]$. Therefore the total costs in $[1, w+S]$ for small $\epsilon$ are $cost(RHC) = \Lambda(w+S)e_0 + \Lambda S\beta_0 + O(\epsilon)$ and $cost(OPT) = \Lambda(w+S)e_0 + \Lambda\beta_0 + O(\epsilon)$. Therefore,

$$\frac{cost(RHC)}{cost(OPT)} = 1 + \frac{(S-1)\beta_0}{(w+S)e_0 + \beta_0} + O(\epsilon).$$

For $S \gg w$ and $Se_0 \gg \beta_0$ and small $\epsilon$, this ratio will approach $1+\beta_0/e_0$, which implies the result. $\square$

# Chapter 5

# Smoothed Online Convex Optimization

In Chapter 3 and Chapter 4 we have seen that many data center management problems can be formalized as optimization problems whose objective has two components: one is the operating cost given the decision in each timeslot, the other is the switching cost for changing the decision between timeslots. Intuitively, we seek to minimize the sum of a sequence of convex functions when "smooth" solutions are preferred, i.e., it is a *smoothed online convex optimization* (SOCO) problem. This class of problems also has many other important applications outside of data center area, including video streaming [66], automatically switched optical networks (ASONs) [126], power generation with dynamic demand [69] and so on. Given the fact that so many applications can be captured by SOCO, without a doubt similar online problems have been studied in different research communities such as online algorithm community, online learning community and control community. For example, in an *online convex optimization* (OCO) problem in the online learning community, a learner interacts with an environment in a sequence of rounds. During each round $t$: (i) the learner must choose an action $x^t$ from a convex decision space $F$; (ii) the environment then reveals a cost convex function $c^t$, and (iii) the learner experiences cost $c^t(x^t)$. The goal is to design learning algorithms that minimize the cost experienced over a (long) horizon of $T$ rounds.

The SOCO model we study in this chapter is a generalization of online convex optimization. The only change in SOCO compared to OCO is that the cost experienced by the learner each round is now $c^t(x^t) + \|x^t - x^{t-1}\|$, where $\|\cdot\|$ is a seminorm.[1] That is, the learner experiences an additional "smoothing cost" or "switching cost" associated with changing the action.

Many applications typically modeled using online convex optimization have, in reality, some cost associated with a change of action. For example, switching costs in the $k$-armed bandit setting have received considerable attention [9, 54]. Further, SOCO has applications even in contexts where there are no costs associated with switching actions. For example, if there is concept drift in a penalized

[1] Recall that a seminorm satisfies the axioms of a norm except that $\|x\| = 0$ does not imply $x = 0$.

estimation problem, it is natural to make use of a regularizer (switching cost) term in order to control the speed of the drift of the estimator.

Though the precise formulation of SOCO does not appear to have been studied previously, there are two large bodies of literature on closely related problems: (i) the online convex optimization (OCO) literature within the machine learning community, e.g., [128, 60], and (ii) the metrical task system (MTS) literature within the algorithms community, e.g., [29, 89]. We discuss these literatures in detail in Section 5.2. While there are several differences between the formulations in the two communities, a notable difference is that they focus on different performance metrics.

In the OCO literature, the goal is typically to minimize the ***regret***, which is the difference between the cost of the algorithm and the cost of the offline optimal static solution. In this context, a number of algorithms have been shown to provide sub-linear regret (also called "no regret"). For example, online gradient descent can achieve $O(\sqrt{T})$-regret [128]. Though such guarantees are proven only in the absence of switching costs, we show in Section 5.2.1 that the same regret bound also holds for SOCO.

In the MTS literature, the goal is typically to minimize the ***competitive ratio***, which is the maximum ratio between the cost of the algorithm and the cost of the offline optimal (dynamic) solution. In this setting, most results tend to be "negative", e.g., when $c^t$ are arbitrary, for any metric space the competitive ratio of an MTS algorithm with states chosen from that space grows without bound as the number of states grows [29, 25]. However, these results still yield competitive ratios that do not increase with the number of tasks, i.e., with time. Throughout, we will neglect dependence of the competitive ratio on the number of states, and describe the competitive ratio as "constant" if it does not grow with time. Notice that positive results are possible when the cost function and decision space are convex, as shown in Chapter 3 and Chapter 4.

Interestingly, the focus on different performance metrics in the OCO and MTS communities has led the communities to develop quite different styles of algorithms. The differences between the algorithms is highlighted by the fact that *all algorithms developed in the OCO community have poor competitive ratio and all algorithms developed in the MTS community have poor regret.*

However, it is natural to seek algorithms with both low regret and low competitive ratio. In learning theory, doing well for both corresponds to being able to learn both static and dynamic concepts well. In the design of a dynamic controller, low regret shows that the control is not more risky than static control, whereas low competitive ratio shows that the control is nearly as good as the best dynamic controller.

The first to connect the two metrics were [22], who treat the special case where the switching costs are a fixed constant, instead of a norm. In this context, they show how to translate bounds on regret to bounds on the competitive ratio, and vice versa. A recent breakthrough was made by [31] who use a primal-dual approach to develop an algorithm that performs well for the "$\alpha$-unfair

competitive ratio", which is a hybrid of the competitive ratio and regret that provides comparison to the dynamic optimal when $\alpha = 1$ and to the static optimal when $\alpha = \infty$ (see Section 5.1). Their algorithm was not shown to perform well *simultaneously* for regret and the competitive ratio, but the result highlights the feasibility of unified approaches for algorithm design across competitive ratio and regret.[2]

This chapter explores the relationship between minimizing regret and minimizing the competitive ratio. To this end, we seek to answer the following question: "Can an algorithm simultaneously achieve both a constant competitive ratio and a sub-linear regret?"

To answer this question, we show that *there is a fundamental incompatibility between regret and competitive ratio* — no algorithm can maintain both sublinear regret and a constant competitive ratio (Theorems 5.1, 5.2, and 5.3). This "incompatibility" does not stem from a pathological example: it holds even for the simple case when $c^t$ is linear and $x^t$ is scalar. Further, it holds for both deterministic and randomized algorithms and also when the $\alpha$-unfair competitive ratio is considered.

Though providing both sub-linear regret and a constant competitive ratio is impossible, we show that it is possible to "nearly" achieve this goal. We present an algorithm, *"randomly biased greedy" (RBG), which achieves a competitive ratio of* $(1+\gamma)$ *while maintaining* $O(\max\{T/\gamma, \gamma\})$ *regret* for $\gamma \geq 1$ on one-dimensional action spaces. If $\gamma$ can be chosen as a function of $T$, then this algorithm can balance between regret and the competitive ratio. For example, it can achieve sub-linear regret while having an arbitrarily slowly growing competitive ratio, or it can achieve $O(\sqrt{T})$ regret while maintaining an $O(\sqrt{T})$ competitive ratio. Note that, unlike the scheme of [31], this algorithm has a finite competitive ratio on continuous action spaces and provides a *simultaneous* guarantee on both regret and the competitive ratio.

## 5.1 Problem formulation

An instance of smoothed online convex optimization (SOCO) consists of a convex decision/action space $F \subseteq (\mathbb{R}^+)^n$ and a sequence of cost functions $\{c^1, c^2, \dots\}$, where each $c^t : F \to \mathbb{R}^+$. At each time $t$, a learner/algorithm chooses an action vector $x^t \in F$ and the environment chooses a cost function $c^t$. Define the *$\alpha$-penalized cost with lookahead $i$* for the sequence $\dots, x^t, c^t, x^{t+1}, c^{t+1}, \dots$ to be

$$C_i^\alpha(A, T) = \mathbb{E}\left[\sum_{t=1}^{T} c^t(x^{t+i}) + \alpha\|x^{t+i} - x^{t+i-1}\|\right],$$

where $x^1, \dots, x^T$ are the decisions of algorithm $A$, the initial action is $x^i = 0$ without loss of generality, the expectation is over randomness in the algorithm, and $\|\cdot\|$ is a seminorm on $\mathbb{R}^n$. The

[2]There is also work on achieving simultaneous guarantees with respect to the static and dynamic optima in other settings, e.g., decision making on lists and trees [24], and there have been applications of algorithmic approaches from machine learning to MTS [23, 2].

parameter $T$ will usually be suppressed.

In the OCO and MTS literatures the learners pay different special cases of this cost. In OCO the algorithm "plays first" giving a 0-step lookahead and switching costs are ignored, yielding $C_0^0$. In MTS the environment plays first giving the algorithm 1-step lookahead ($i = 1$), and uses $\alpha = 1$, yielding $C_1^1$. Note that we sometimes omit the superscript when $\alpha = 1$, and the subscript when $i = 0$.

One can relate the MTS and OCO costs by relating $C_i^\alpha$ to $C_{i-1}^\alpha$, as done by [22] and [31]. The penalty due to not having lookahead is

$$c^t(x^t) - c^t(x^{t+1}) \leq \nabla c^t(x^t)(x^t - x^{t+1}) \leq \|\nabla c^t(x^t)\|_2 \cdot \|x^t - x^{t+1}\|_2, \tag{5.1}$$

where $\|\cdot\|_2$ is the Euclidean norm. We adopt the assumption, common in the OCO literature, that $\|\nabla c^t(\cdot)\|_2$ are bounded on a given instance; which thus bounds the difference between the costs of MTS and OCO (with switching cost), $C_1$ and $C_0$.

**Performance metrics.** The performance of a SOCO algorithm is typically evaluated by comparing its cost to that of an offline "optimal" solution, but the communities differ in their choice of benchmark, and whether to compare additively or multiplicatively.

The OCO literature typically compares against the optimal offline *static* action, i.e.,

$$OPT_s = \min_{x \in F} \sum_{t=1}^{T} c^t(x),$$

and evaluates the ***regret***, defined as the (additive) difference between the algorithm's cost and that of the optimal static action vector. Specifically, the regret $R_i(A)$ of Algorithm $A$ with lookahead $i$ on instances $\mathfrak{C}$, is less than $\rho(T)$ if for any sequence of cost functions $(c^1, \ldots, c^T) \in \mathfrak{C}^T$,

$$C_i^0(A) - OPT_s \leq \rho(T) \tag{5.2}$$

Note that for any problem and any $i \geq 1$ there exists an algorithm $A$ for which $R_i(A)$ is non-positive; however, an algorithm that is not designed specifically to minimize regret may have $R_i(A) > 0$.

This traditional definition of regret omits switching costs and lookahead (i.e., $R_0(A)$). One can generalize regret to define $R_i'(A)$, by replacing $C_i^0(A)$ with $C_i^1(A)$ in (5.2). Specifically, $R_i'(A)$ is less than $\rho(T)$ if for any sequence of cost functions $(c^1, \ldots, c^T) \in \mathfrak{C}^T$,

$$C_i^1(A) - OPT_s \leq \rho(T) \tag{5.3}$$

Except where noted, we use the set $\mathfrak{C}^1$ of sequences of convex functions mapping $(\mathbb{R}^+)^n$ to $\mathbb{R}^+$ with

(sub)gradient uniformly bounded over the sequence. Note that we do not require differentiability; throughout this chapter, references to gradients can be read as references to subgradients.

The MTS literature typically compares against the optimal offline (dynamic) solution,

$$OPT_d = \min_{x \in F^T} \sum_{t=1}^{T} c^t(x^t) + \|x^t - x^{t-1}\|,$$

and evaluates the ***competitive ratio***. The cost most commonly considered is $C_1$. More generally, we say the competitive ratio with lookahead $i$, denoted by $CR_i(A)$, is $\rho(T)$ if for any sequence of cost functions $(c^1, \ldots, c^T) \in \mathfrak{C}^T$

$$C_i(A) \leq \rho(T) OPT_d + O(1). \tag{5.4}$$

**Bridging competitiveness and regret.** Many hybrid benchmarks have been proposed to bridge static and dynamic comparisons. For example, adaptive-regret [61] is the maximum regret over any interval, where the "static" optimum can differ for different intervals, and internal regret [26] compares the online policy against a simple perturbation of that policy. We adopt the static-dynamic hybrid proposed in the most closely related literature [25, 22, 31], the ***$\alpha$-unfair competitive ratio***, which we denote by $CR_i^\alpha(A)$ for lookahead $i$. For $\alpha \geq 1$, $CR_i^\alpha(A)$ is $\rho(T)$ if (5.4) holds with $OPT_d$ replaced by

$$OPT_d^\alpha = \min_{x \in F^T} \sum_{t=1}^{T} c^t(x^t) + \alpha\|x^t - x^{t-1}\|.$$

Specifically, the $\alpha$-unfair competitive ratio with lookahead $i$, $CR_i^\alpha(A)$, is $\rho(T)$ if for any sequence of cost functions $(c^1, \ldots, c^T) \in \mathfrak{C}^T$

$$C_i(A) \leq \rho(T) OPT_d^\alpha + O(1). \tag{5.5}$$

For $\alpha = 1$, $OPT_d^\alpha$ is the dynamic optimum; as $\alpha \to \infty$, $OPT_d^\alpha$ approaches the static optimum.

To bridge the additive versus multiplicative comparisons used in the two literatures, we define the ***competitive difference***. The $\alpha$-unfair competitive difference with lookahead $i$ on instances $\mathfrak{C}$, $CD_i^\alpha(A)$, is $\rho(T)$ if for any sequence of cost functions $(c^1, \ldots, c^T) \in \mathfrak{C}^T$,

$$C_i(A) - OPT_d^\alpha \leq \rho(T). \tag{5.6}$$

This measure allows for a smooth transition between regret (when $\alpha$ is large enough) and an additive version of the competitive ratio when $\alpha = 1$.

## 5.2 Background

In the following, we briefly discuss related work on both online convex optimization and metrical task systems, to provide context for the results in this chapter.

### 5.2.1 Online convex optimization

The OCO problem has a rich history and a wide range of important applications. In computer science, OCO is perhaps most associated with the $k$-experts problem [62, 86], a discrete-action version of online optimization wherein at each round $t$ the learning algorithm must choose a number between 1 and $k$, which can be viewed as following the advice of one of $k$ "experts." However, OCO also has a long history in other areas, such as portfolio management [41, 33].

The identifying features of the OCO formulation are that (i) the typical performance metric considered is regret, (ii) switching costs are not considered, and (iii) the learner must act before the environment reveals the cost function. In our notation, the cost function in the OCO literature is $C^0(A)$ and the performance metric is $R_0(A)$. Following [128] and [60], the typical assumptions are that the decision space $F$ is non-empty, bounded and closed, and that the Euclidean norms of gradients $\|\nabla c^t(\cdot)\|_2$ are also bounded.

In this setting, a number of algorithms have been shown to achieve "no regret", i.e., sublinear regret, $R_0(A) = o(T)$. An important example is *online gradient descent* (OGD), which is parameterized by learning rates $\eta_t$. OGD works as follows.

**Algorithm 5.1** (Online gradient descent, OGD)**.** *Select arbitrary $x^1 \in F$. In time step $t \geq 1$, select $x^{t+1} = P(x^t - \eta_t \nabla c^t(x^t))$, where $P(y) = \arg\min_{x \in F} \|x - y\|_2$ is the projection under the Euclidean norm.*

With appropriate learning rates $\eta_t$, OGD achieves sub-linear regret for OCO. In particular, the variant of [128] uses $\eta_t = \Theta(1/\sqrt{t})$ and obtains $O(\sqrt{T})$-regret. Tighter bounds are possible in restricted settings. [60] achieves $O(\log T)$ regret by choosing $\eta_t = 1/(\gamma t)$ for settings when the cost function additionally is twice differentiable and has minimal curvature, i.e., $\nabla^2 c^t(x) - \gamma I_n$ is positive semidefinite for all $x$ and $t$, where $I_n$ is the identity matrix of size $n$. In addition to algorithms based on gradient descent, more recent algorithms such as online Newton step and follow the approximate leader [60] also attain $O(\log T)$-regret bounds for a class of cost functions.

None of the work discussed above considers switching costs. To extend the literature discussed above from OCO to SOCO, we need to track the switching costs incurred by the algorithms. This leads to the following straightforward result, proven in Appendix 5.A.

**Proposition 5.1.** *Consider an online gradient descent algorithm $A$ on a finite dimensional space with learning rates such that $\sum_{t=1}^{T} \eta_t = O(\rho_1(T))$. If $R_0(A) = O(\rho_2(T))$, then $R_0'(A) = O(\rho_1(T) +$*

$\rho_2(T))$.

Interestingly, the choices of $\eta_t$ used by the algorithms designed for OCO also turn out to be good choices to control the switching costs of the algorithms. The algorithms of [128] and [60], which use $\eta_t = 1/\sqrt{t}$ and $\eta_t = 1/(\gamma t)$, are still $O(\sqrt{T})$-regret and $O(\log T)$-regret respectively when switching costs are considered, since in these cases $\rho_1(T) = O(\rho_2(T))$. Note that a similar result can be obtained for online Newton step [60].

Importantly, though the regret of OGD algorithms is sublinear, it can easily be shown that the competitive ratio of these algorithms is unbounded.

### 5.2.2 Metrical task systems

Like OCO, MTS also has a rich history and a wide range of important applications. Historically, MTS is perhaps most associated with the $k$-server problem [40]. In this problem, there are $k$ servers, each in some state, and a sequence of requests is incrementally revealed. To serve a request, the system must move one of the servers to the state necessary to serve the request, which incurs a cost that depends on the source and destination states.

The formulation of SOCO in Section 5.1 is actually, in many ways, a special case of the most general MTS formulation. In general, the MTS formulation differs in that (i) the cost functions $c^t$ are not assumed to be convex, (ii) the decision space is typically assumed to be discrete and is not necessarily embedded in a vector space, and (iii) the switching cost is an arbitrary metric $d(x^t, x^{t-1})$ rather than a seminorm $\|x^t - x^{t-1}\|$. In this context, the cost function studied by MTS is typically $C_1$ and the performance metric of interest is the competitive ratio, specifically $CR_1(A)$, although the $\alpha$-unfair competitive ratio $CR_1^\alpha$ also receives attention.

The weakening of the assumptions on the cost functions, and the fact that the competitive ratio uses the dynamic optimum as the benchmark, means that most of the results in the MTS setting are "negative" when compared with those for OCO. In particular, it has been proven that, given an arbitrary metric decision space of size $n$, any deterministic algorithm must be $\Omega(n)$-competitive [29]. Further, any randomized algorithm must be $\Omega(\sqrt{\log n / \log\log n})$-competitive [25].

These results motivate imposing additional structure on the cost functions in order to attain positive results. For example, it is commonly assumed that the metric is the uniform metric, in which $d(x, y)$ is equal for all $x \neq y$; that assumption was made by [22] in a study of the tradeoff between competitive ratio and regret. For comparison with OCO, an alternative natural restriction is to impose convexity assumptions on the cost function and the decision space, as done in this chapter.

Upon restricting $c^t$ to be convex, $F$ to be convex, and $\|\cdot\|$ to be a semi-norm, the MTS formulation becomes quite similar to the SOCO formulation. This restricted class has been the focus of Chapter

3 and Chapter 4 in this thesis and we know that positive results are possible. For example, when $F$ is a one-dimensional normed space,[3] lazy capacity provisioning (LCP) is 3-competitive.

Importantly, though the algorithms described above provide constant competitive ratios, in all cases it is easy to see that the regret of these algorithms is linear.

## 5.3 The incompatibility of regret and the competitive ratio

As noted in the introduction, there is considerable motivation to perform well for regret and competitive ratio simultaneously. See also [25, 22, 31, 61, 26]. None of the algorithms discussed so far achieves this goal. For example, online gradient descent has sublinear regret but its competitive ratio is infinite. Similarly, lazy capacity provisioning is 3-competitive but has linear regret.

This is no accident. We show below that the two goals are fundamentally incompatible: any algorithm that has sublinear regret for OCO necessarily has an infinite competitive ratio for MTS; and any algorithm that has a constant competitive ratio for MTS necessarily has at least linear regret for OCO. Further, our results give lower bounds on the simultaneous guarantees that are possible.

In discussing this "incompatibility", there are a number of subtleties as a result of the differences in formulation between the OCO literature, where regret is the focus, and the MTS literature, where competitive ratio is the focus. In particular, there are four key differences which are important to highlight: (i) OCO uses lookahead $i = 0$ MTS uses $i = 1$; (ii) OCO does not consider switching costs ($\alpha = 0$) while MTS does ($\alpha = 1$); (iii) regret uses an additive comparison while the competitive ratio uses a multiplicative comparison; and (iv) regret compares to the static optimal while competitive ratio compares to the dynamic optimal. Note that the first two are intrinsic to the costs, while the latter are intrinsic to the performance metric. The following teases apart which of these differences create incompatibility and which do not. In particular, we prove that (i) and (iv) each create incompatibilities.

Our first result in this section states that there is an incompatibility between regret in the OCO setting and the competitive ratio in the MTS setting, i.e., between the two most commonly studied measures $R_0(A)$ and $CR_1(A)$. Naturally, the incompatibility remains if switching costs are added to regret, i.e., $R'_0(A)$ is considered. Further, the incompatibility remains when the competitive difference is considered, and so both the comparison with the static optimal and the dynamic optimal are additive. In fact, the incompatibility remains even when the $\alpha$-unfair competitive ratio/difference is considered. Perhaps most surprisingly, the incompatibility remains when there is lookahead, i.e., when $C_i$ and $C_{i+1}$ are considered.

[3] We need only consider the absolute value norm, since for every seminorm $\|\cdot\|$ on $\mathbb{R}$, $\|x\| = \|1\||x|$.

**Theorem 5.1.** *Consider an arbitrary seminorm $\|\cdot\|$ on $\mathbb{R}^n$, constants $\gamma > 0$, $\alpha \geq 1$ and $i \in \mathbb{N}$. There is a $\mathfrak{C}$ containing a single sequence of cost functions such that, for all deterministic and randomized algorithms $A$, either $R_i(A) = \Omega(T)$ or for large enough $T$, both $CR^{\alpha}_{i+1}(A) \geq \gamma$ and $CD^{\alpha}_{i+1}(A) \geq \gamma T$.*

The incompatibility arises even in "simple" instances; the proof of Theorem 5.1 uses linear cost functions and a one-dimensional decision space, and the construction of the cost functions does not depend on $T$ or $A$.

The cost functions used by regret and the competitive ratio in Theorem 5.1 are "off by one", motivated by the different settings in OCO and MTS. However, the following shows that parallel results also hold when the cost functions are not "off by one", i.e., for $R_0(A)$ vs. $CR^{\alpha}_0(A)$ and $R'_1(A)$ vs. $CR^{\alpha}_1(A)$.

**Theorem 5.2.** *Consider an arbitrary seminorm $\|\cdot\|$ on $\mathbb{R}^n$, constants $\gamma > 0$ and $\alpha \geq 1$, and a deterministic or randomized online algorithm $A$. There is a $\mathfrak{C}$ containing two cost functions such that either $R_0(A) = \Omega(T)$ or, for large enough $T$, both $CR^{\alpha}_0(A) \geq \gamma$ and $CD^{\alpha}_0(A) \geq \gamma T$.*

**Theorem 5.3.** *Consider an arbitrary norm $\|\cdot\|$ on $\mathbb{R}^n$. There is a $\mathfrak{C}$ containing two cost functions such that, for any constants $\gamma > 0$ and $\alpha \geq 1$ and any deterministic or randomized online algorithm $A$, either $R'_1(A) = \Omega(T)$, or for large enough $T$, $CR^{\alpha}_1(A) \geq \gamma$.*

The impact of these results can be stark. It is impossible for an algorithm to learn static concepts with sublinear regret in the OCO setting, while having a constant competitive ratio for learning dynamic concepts in the MTS setting. More strikingly, in control theory, any dynamic controller that has a constant competitive ratio must have at least linear regret, and so there are cases where it does much worse than the best static controller. Thus, one cannot simultaneously guarantee the dynamic policy is always as good as the best static policy and is nearly as good as the optimal dynamic policy.

Theorem 5.3 is perhaps the most interesting of these results. Theorem 5.1 is due to seeking to minimize different cost functions ($c^t$ and $c^{t+1}$), while Theorem 5.2 is due to the hardness of attaining a small $CR^{\alpha}_0$, i.e., of mimicking the dynamic optimum without 1-step lookahead. In contrast, for Theorem 5.3, algorithms exist with strong performance guarantees for each measure individually, and the measures are aligned in time. However, Theorem 5.3 must consider the (nonstandard) notion of regret that includes switching costs ($R'$), since otherwise the problem is trivial.

We now prove the results above. We use one-dimensional examples; however these examples can easily be embedded into higher dimensions if desired. We show proofs only for competitive ratio; the proofs for competitive difference are similar.

Let $\bar{\alpha} = \max(1, \|\alpha\|)$. Given $a > 0$ and $b \geq 0$, define two possible cost functions on $F = [0, 1/\bar{\alpha}]$: $f^{\alpha}_1(x) = b + ax\bar{\alpha}$ and $f^{\alpha}_2(x) = b + a(1 - x\bar{\alpha})$. These functions are similar to those used by [55] to

study online gradient descent to learn a concept of bounded total variation. To simplify notation, let $D(t) = 1/2 - \mathbb{E}x^t\bar{\alpha}$, and note that $D(t) \in [-1/2, 1/2]$.

**Proof of Theorem 5.1**

To prove Theorem 5.1, we prove the stronger claim that $CR^{\alpha}_{i+1}(A) + R_i(A)/T \geq \gamma$.

Consider a system with costs $c^t = f_1^{\alpha}$ if $t$ is odd and $f_2^{\alpha}$ if $t$ is even. Then $C_i(A) \geq (a/2+b)T + a\sum_{t=1}^{T}(-1)^t D(t+i)$. The static optimum is not worse than the scheme that sets $x^t = 1/(2\bar{\alpha})$ for all $t$, which has total cost no more than $(a/2+b)T + \|1/2\|$. The $\alpha$-unfair dynamic optimum for $C_{i+1}$ is not worse than the scheme that sets $x^t = 0$ if $t$ is odd and $x^t = 1/\bar{\alpha}$ if $t$ is even, which has total $\alpha$-unfair cost at most $(b+1)T$. Hence

$$\begin{aligned} R_i(A) \geq& a\sum_{t=1}^{T}(-1)^t D(t+i) - \|1/2\| \\ CR^{\alpha}_{i+1}(A) \geq& \frac{(a/2+b)T + a\sum_{t=1}^{T}(-1)^t D(t+i+1)}{(b+1)T} \end{aligned}$$

Thus, since $D(t) \in [-1/2, 1/2]$,

$$\begin{aligned} &(b+1)T(CR^{\alpha}_{i+1}(A) + R_i(A)/T) + (b+1)\|1/2\| - (a/2+b)T \\ \geq& a\sum_{t=1}^{T}(-1)^t(D(t+i+1) + (b+1)D(t+i)) \\ =& ab\sum_{t=1}^{T}(-1)^t D(t+i) - a\left(D(i+1) + (-1)^T D(T+i+1)\right) \geq -abT/2 - a \end{aligned}$$

To establish the claim, it is then sufficient that $(a/2+b)T - (b+1)\|1/2\| - abT/2 - a \geq \gamma T(b+1)$. For $b = 1/2$ and $a = 30\gamma + 2 + \|6\|$, this holds for $T \geq 5$.

**Proof of Theorem 5.2**

To prove Theorem 5.2, we again prove the stronger claim $CR^{\alpha}_0(A) + R_0(A)/T \geq \gamma$.

Consider the cost function sequence with $c^t(\cdot) = f_2^0$ for $\mathbb{E}x^t \leq 1/2$ and $c^t(\cdot) = f_1^0$ otherwise, on decision space $[0,1]$, where $x^t$ is the (random) choice of the algorithm at round $t$. Here the expectation is taken over the marginal distribution of $x^t$ conditioned on $c_1, \ldots, c_{t-1}$, averaging out the dependence on the realizations of $x_1, \ldots, x_{t-1}$. Notice that this sequence can be constructed by an oblivious adversary before the execution of the algorithm.

The following lemma is proven in Appendix 5.B.

**Lemma 5.1.** *Given any algorithm, the sequence of cost functions chosen by the above oblivious*

*adversary makes*

$$R_0(A), R_0'(A) \geq a \sum_{t=1}^{T} |1/2 - \mathbb{E}x^t| - \|1/2\|, \tag{5.7}$$

$$CR_0^\alpha(A) \geq \frac{(a/2+b)T + a\sum_{t=1}^{T} |1/2 - \mathbb{E}x^t|}{(b+\|\alpha\|)T} \tag{5.8}$$

From (5.7) and (5.8) in Lemma 5.1, we have $CR_0^\alpha(A) + R_0(A)/T \geq \frac{(a/2+b)T}{(b+\|\alpha\|)T} - \frac{\|1/2\|}{T}$. For $a > 2\gamma(b + \|\alpha\|)$, the right hand side is bigger than $\gamma$ for sufficiently large $T$, which establishes the claim.

**Proof of Theorem 5.3**

Let $a = \|1\|/2$ and $b = 0$. Let $M = 4\alpha\gamma\|1\|/a = 8\alpha\gamma$. For $T \gg M$, divide $[1, T]$ into segments of length $3M$. For the last $2M$ of each segment, set $c^t = f_1^\alpha$. This ensures that the static optimal solution is $x = 0$. Moreover, if, for all $t$ in the first $M$ time steps, $c^t$ is either $f_1^\alpha$ or $f_2^\alpha$, then the optimal dynamic solution is also $x^t = 0$ for the last $2M$ time steps.

Consider a solution on which each segment has non-negative regret. Then to obtain sublinear regret, for any positive threshold $\epsilon$ at least $T/(3M) - o(T)$ of these segments must have regret below $\epsilon\|1\|$. We will then show that these segments must have high competitive ratio. To make this more formal, consider (w.l.o.g.) the single segment $[1, 3M]$.

Let $\tilde{c}$ be such that $\tilde{c}^t = f_2^\alpha$ for all $t \in [1, M]$ and $\tilde{c}^t = f_1^\alpha$ for $t > M$. Then the optimal dynamic solution on $[1, 3M]$ is $x_d^t = \mathbf{1}_{t \leq M}$, which has total cost $2\alpha\|1\|$.

The following lemma is proven in Appendix 5.C.

**Lemma 5.2.** *For any $\delta \in (0, 1)$ and integer $\tau > 0$, there exists an $\epsilon(\delta, \tau) > 0$ such that, if $c^t = f_2^\alpha$ for all $1 \leq t \leq \tau$ and $x^t > \delta$ for any $1 \leq t \leq \tau$, then there exists an $m \leq \tau$ such that $C_1(x, m) - C_1(OPT_s, m) > \epsilon(\delta, \tau)\|1\|$.*

Choose $\delta = 1/[5\max(1, \|\alpha\|)] \in (0, 1)$ such that, for any decisions such that $x^t < \delta$ for all $t \in [1, M]$, the cost of $x$ under $\tilde{c}$ is at least $3\alpha\gamma\|1\|$. Let the adversary choose a $c$ on this segment such that $c^t = f_2^\alpha$ until (a) the first time $t_0 < M$ that the algorithm's solution $x$ satisfies $C_1(x, t_0) - C_1(OPT_s, t_0) > \epsilon(\delta, M)\|1\|$, or (b) $t = M$. After this, it chooses $c^t = f_1^\alpha$.

In case (a), $C_1(x, 3M) - C_1(OPT_s, 3M) > \epsilon(\delta, M)\|1\|$, since $OPT_s$ incurs no cost after $t_0$. Moreover $C_1(x, 3M) \geq C_1(OPT_d, 3M)$.

In case (b), $C_1(x, 3M)/C_1(OPT_d, 3M) \geq 3\alpha\gamma\|1\|/(2\alpha\|1\|) = 3\gamma/2$.

To complete the argument, consider all segments. Let $g(T)$ be the number of segments for which case (a) occurs. The regret then satisfies

$$R_1'(A) \geq \epsilon(\delta, M)\|1\|g(T).$$

Similarly, the ratio of the total cost to that of the optimum is at least

$$\frac{C_1(x,T)}{C_1(OPT_d,T)} \geq \frac{[T/(3M)-g(T)]3\alpha\gamma\|1\|}{[T/(3M)]2\alpha\|1\|} = \frac{3}{2}\gamma\left(1-\frac{3Mg(T)}{T}\right).$$

If $g(T)=\Omega(T)$ then then $R_1'(A)=\Omega(T)$. Conversely, if $g(T)=o(T)$ then for sufficiently large $T$, $3Mg(T)/T<1/3$ and so $CR_1^{\alpha}(A)>\gamma$.

## 5.4 Balancing regret and the competitive ratio

Given the above incompatibility, it is necessary to reevaluate the goals for algorithm design. In particular, it is natural now to seek tradeoffs such as being able to obtain $\epsilon T$ regret for arbitrarily small $\epsilon$ while remaining $O(1)$-competitive, or being $\log\log T$-competitive while retaining sublinear regret.

To this end, in the following we present a novel algorithm, randomly biased greedy (RBG), which can achieve simultaneous bounds on regret $R_0'$ and competitive ratio $CR_1$, when the decision space $F$ is one-dimensional. The one-dimensional setting is the natural starting point for seeking such a tradeoff given that the proofs of the incompatibility results all focus on one-dimensional examples and that the one-dimensional case is of practical significance like the data center management in Chapter 3. The algorithm takes a norm $N$ as its input:

**Algorithm 5.2** (Randomly biased greedy, RBG($N$))**.**
*Given a norm $N$, define $w^0(x)=N(x)$ for all $x$ and $w^t(x)=\min_y\{w^{t-1}(y)+c^t(y)+N(x-y)\}$. Generate a random number $r$ uniformly in $(-1,1)$. For each time step $t$, go to the state $x^t$ which minimizes $Y^t(x^t)=w^{t-1}(x^t)+rN(x^t)$.*

RBG is motivated by [40], and makes very limited use of randomness – it parameterizes its "bias" using a single random $r\in(-1,1)$. It then chooses actions to greedily minimize its "work function" $w^t(x)$.

As stated, RBG performs well for the $\alpha$-unfair competitive ratio, but performs poorly for the regret. Theorem 5.4 will show that RBG($\|\cdot\|$) is 2-competitive,[4] and hence has at best linear regret. However, the key idea behind balancing regret and competitive ratio is to run RBG with a "larger" norm to encourage its actions to change less. This can make the coefficient of regret arbitrarily small, at the expense of a larger (but still constant) competitive ratio.

**Theorem 5.4.** *For a SOCO problem in a one-dimensional normed space $\|\cdot\|$, running RBG($N$) with a one-dimensional norm having $N(1)=\theta\|1\|$ as input (where $\theta\geq 1$) attains an $\alpha$-unfair competitive ratio $CR_1^{\alpha}$ of $(1+\theta)/\min\{\theta,\alpha\}$ and a regret $R_0'$ of $O(\max\{T/\theta,\theta\})$.*

[4]Note that this improves the best known competitive ratio for this setting from 3 (achieved by lazy capacity provisioning) to 2.

Note that Theorem 5.4 holds for the usual metrics of MTS and OCO, which are the "most incompatible" case since the cost functions are mismatched (cf. Theorem 5.1). Thus, the conclusion of Theorem 5.4 still holds when $R_0$ or $R_1$ is considered in place of $R_0'$.

The best $CR_1^\alpha$, $1 + 1/\alpha$, achieved by RBG is obtained with $N(\cdot) = \alpha\|\cdot\|$. However, choosing $N(\cdot) = \|\cdot\|/\epsilon$ for arbitrarily small $\epsilon$, gives $\epsilon T$-regret, albeit larger $CR_1^\alpha$. Similarly, if $T$ is known in advance, choosing $N(1) = \theta(T)$ for some increasing function achieves an $O(\theta(T))$ $\alpha$-unfair competitive ratio and $O(\max\{T/\theta(T), \theta(T)\})$ regret; taking $\theta(T) = O(\sqrt{T})$ gives $O(\sqrt{T})$ regret, which is optimal for arbitrary convex costs [128]. If $T$ is not known in advance, $N(1)$ can increase in $t$, and bounds similar to those in Theorem 5.4 still hold.

**Proof of Theorem 5.4**

To prove Theorem 5.4, we derive a more general tool for designing algorithms that simultaneously balance regret and the $\alpha$-unfair competitive ratio. In particular, for any algorithm $A$, let the operating cost be $OC(A) = \sum_{t=1}^T c^t(x^{t+1})$ and the switching cost be $SC(A) = \sum_{t=1}^T \|x^{t+1} - x^t\|$, so that $C_1(A) = OC(A) + SC(A)$. Define $OPT_N$ to be the dynamic optimal solution under the norm $N(1) = \theta\|1\|$ ($\theta \geq 1$) with $\alpha = 1$. The following lemma is proven in Appendix 5.D.

**Lemma 5.3.** *Consider a one-dimensional SOCO problem with norm $\|\cdot\|$ and an online algorithm $A$ which, when run with norm $N$, satisfies $OC(A(N)) \leq OPT_N + O(1)$ along with $SC(A(N)) \leq \beta OPT_N + O(1)$ with $\beta = O(1)$. Fix a norm $N$ such that $N(1) = \theta\|1\|$ with $\theta \geq 1$. Then $A(N)$ has $\alpha$-unfair competitive ratio $CR_1^\alpha(A(N)) = (1+\beta)\max\{\frac{\theta}{\alpha}, 1\}$ and regret $R_0'(A(N)) = O(\max\{\beta T, (1+\beta)\theta\})$ for the original SOCO problem with norm $\|\cdot\|$.*

Theorem 5.4 then follows from the following lemmas, proven in Appendices 5.E and 5.F.

**Lemma 5.4.** *Given a SOCO problem with norm $\|\cdot\|$, $\mathbb{E}OC(RBG(N)) \leq OPT_N$.*

**Lemma 5.5.** *Given a one-dimensional SOCO problem with norm $\|\cdot\|$,*
*$\mathbb{E}SC(RBG(N)) \leq OPT_N/\theta$ with probability 1.*

## 5.5 Concluding remarks

This chapter studies the relationship between regret and competitive ratio when applied to the class of SOCO problems. It shows that these metrics, from the learning and algorithms communities respectively, are fundamentally incompatible, in the sense that algorithms with sublinear regret must have infinite competitive ratio, and those with constant competitive ratio have at least linear regret. Thus, the choice of performance measure significantly affects the style of algorithm designed. It also introduces a generic approach for balancing these competing metrics, exemplified by a specific algorithm, RBG.

There are a number of interesting directions that this work motivates. In particular, the SOCO formulation is still under-explored, and many variations of the formulation discussed here are still not understood. For example, is it possible to tradeoff between regret and the competitive ratio in bandit versions of SOCO? More generally, the message from this chapter is that regret and the competitive ratio are incompatible within the formulation of SOCO. It is quite interesting to try to understand how generally this holds, for example, under the settings where the cost functions are random instead of adversarial, e.g., variations of SOCO such as $k$-armed bandit problems with switching costs.

## Appendix 5.A Proof of Proposition 5.1

Recall that, by assumption, $\|\nabla c^t(\cdot)\|_2$ is bounded. So, let us define $D$ such that $\|\nabla c^t(\cdot)\|_2 \leq D$. Next, due to the fact that all norms are equivalent in a finite dimensional space, there exist $m, M > 0$ such that for every $x$, $m\|x\|_a \leq \|x\|_b \leq M\|x\|_a$. Combining these facts, we can bound the switching cost incurred by an OGD algorithm as follows:

$$\sum_{t=1}^{T} \|x^t - x^{t-1}\| \leq M \sum_{t=1}^{T} \|x^t - x^{t-1}\|_2 \leq M \sum_{t=1}^{T} \eta_t \|\nabla c^t(\cdot)\|_2 \leq MD \sum_{t=1}^{T} \eta_t.$$

The second inequality comes from the fact that projection to a convex set under the Euclidean norm is nonexpansive, i.e., $\|P(x) - P(y)\|_2 \leq \|x - y\|_2$. Thus, the switching cost causes an additional regret of $\sum_{t=1}^{T} \eta_t = O(\rho_1(T))$ for the algorithm, completing the proof.

## Appendix 5.B Proof of Lemma 5.1

Recall that the oblivious adversary chooses $c^t(\cdot) = f_2^0$ for $\mathbb{E}x^t \leq 1/2$ and $c^t(\cdot) = f_1^0$ otherwise, where $x^t$ is the (random) choice of the algorithm at round $t$. Therefore,

$$\begin{aligned}
C_0(A) \geq & \mathbb{E} \sum_{t=1}^{T} \begin{cases} a(1-x^t) + b & \text{if } \mathbb{E}x^t \leq 1/2 \\ ax^t + b & \text{otherwise} \end{cases} \\
= & \mathbb{E}bT + a \sum_{t=1}^{T} \big(1/2 + (1/2 - x^t)\text{sgn}(1/2 - \mathbb{E}x^t)\big) \\
= & bT + a \sum_{t=1}^{T} \big(1/2 + (1/2 - \mathbb{E}x^t)\text{sgn}(1/2 - \mathbb{E}x^t)\big) \\
= & (a/2 + b)T + a \sum_{t=1}^{T} |1/2 - \mathbb{E}x^t|
\end{aligned}$$

where $\mathrm{sgn}(x) = 1$ if $x > 0$ and $-1$ otherwise. The static optimum is not worse than the scheme that sets $x^t = 1/2$ for all $t$, which has total cost $(a/2 + b)T + \|1/2\|$. This establishes (5.7).

The dynamic scheme which chooses $x^{t+1} = 0$ if $c^t = f_1^0$ and $x^{t+1} = 1$ if $c^t = f_2^0$ has total $\alpha$-unfair cost not more than $(b + \|\alpha\|)T$. This establishes (5.8).

## Appendix 5.C Proof of Lemma 5.2

*Proof.* We prove the contrapositive (that if $C_1(x; m) - C_1(OPT_s, m) \leq \epsilon$ for all $m$ then $x^t < \delta$ for all $t \in [1, \tau]$). We consider the case that $x^t$ are non-decreasing; if not, the switching and operating cost can both be reduced by setting $(x^t)' = \max_{t' \leq t} x^{t'}$.

Note that $OPT_s$ sets $x^t = 0$ for all $t$, whence $C_1(OPT_s, m) = am$, and that

$$C_1(x; m) = x^m \|1\| - a \sum_{i=1}^{m} x^i + am.$$

Thus, we want to show that if $x^m \|1\| - a \sum_{i=1}^{m} x^i \leq \epsilon$ for all $m \leq \tau$ then $x^t < \delta$ for all $t \in [1, \tau]$.

Define $f_i(\cdot)$ inductively by $f_1(y) = 1/(1 - y)$, and

$$f_i(y) = \frac{1}{1 - y} \left( 1 + y \sum_{j=1}^{i-1} f_j(y) \right).$$

If $y < 1$ then $\{f_i(y)\}$ are increasing in $i$.

Notice that $\{f_i\}$ satisfy

$$f_m(y)(1 - y) - y \sum_{i=1}^{m-1} f_i(y) = 1.$$

Expanding the first term gives that for any $\hat{\epsilon}$,

$$\hat{\epsilon} f_m(a/\|1\|) - \frac{a}{\|1\|} \sum_{i=1}^{m} \hat{\epsilon} f_i(a/\|1\|) = \hat{\epsilon}. \tag{5.9}$$

If for some $\hat{\epsilon} > 0$,

$$x^m - \frac{a}{\|1\|} \sum_{j=1}^{m} x^j \leq \hat{\epsilon} \tag{5.10}$$

for all $m \leq \tau$, then by induction $x^i \leq \hat{\epsilon} f_i(a/\|1\|) \leq \hat{\epsilon} f_\tau(a/\|1\|)$ for all $i \leq \tau$, where the last inequality uses the fact that $a < \|1\|$ whence $\{f_i(a/\|1\|)\}$ are increasing in $i$.

The left hand side of (5.10) is $(C_1(x; m) - C_1(OPT_s, m))/\|1\|$. Define $\epsilon = \hat{\epsilon}\|1\| = \delta\|1\|/(2 f_\tau(a/\|1\|))$. Then if $(C_1(x; m) - C_1(OPT_s, m)) \leq \epsilon$ for all $m$, then (5.10) holds for all $m$, whence $x^t \leq \hat{\epsilon} f_\tau(a/\|1\|) = \delta/2 < \delta$ for all $t \in [1, \tau]$. □

# Appendix 5.D Proof of Lemma 5.3

We first prove the $\alpha$-unfair competitive ratio result. Let $\hat{x}^1, \hat{x}^2, \ldots, \hat{x}^T$ denote the actions chosen by algorithm $ALG$ when running on a normed space with $N = \|\cdot\|_{ALG}$ as input. Let $\hat{y}^1, \hat{y}^2, \ldots, \hat{y}^T$ be the actions chosen by the optimal dynamic offline algorithm, which pays $\alpha$ times more for switching costs, on a normed space with $\|\cdot\|$ (i.e., $OPT_d^\alpha$). Similarly, let $\hat{z}^1, \hat{z}^2, \ldots, \hat{z}^T$ be the actions chosen by the optimal solution on a normed space with $\|\cdot\|_{ALG}$, namely $OPT_{\|\cdot\|_{ALG}}$ (without an unfairness cost). Recall that we have $C_1(ALG) = \sum_{t=1}^T c^t(\hat{x}^{t+1}) + \|\hat{x}^{t+1} - \hat{x}^t\|$, $OPT_d^\alpha = \sum_{t=1}^T c^t(\hat{y}^t) + \alpha\|\hat{y}^t - \hat{y}^{t-1}\|$, and $OPT_{\|\cdot\|_{ALG}} = \sum_{t=1}^T c^t(\hat{z}^t) + \|\hat{z}^t - \hat{z}^{t-1}\|_{ALG}$. By the assumptions in our lemma, we know that $C_1(ALG) \le (1+\beta)OPT_{\|\cdot\|_{ALG}} + O(1)$. Moreover,

$$OPT_d^\alpha = \sum_{t=1}^T c^t(\hat{y}^t) + \alpha\|\hat{y}^t - \hat{y}^{t-1}\| \ge \sum_{t=1}^T c^t(\hat{y}^t) + \frac{\alpha}{\theta}\|\hat{y}^t - \hat{y}^{t-1}\|_{ALG} \ge \frac{OPT_{\|\cdot\|_{ALG}}}{\max\{1, \frac{\theta}{\alpha}\}}$$

The first inequality holds since $\|\cdot\|_{ALG} = \theta\|\cdot\|$ with $\theta \ge 1$. Therefore, $C_1(ALG) \le (1+\beta)\max\{1, \frac{\theta}{\alpha}\}OPT_d^\alpha$.

We now prove the regret bound. Let $d_{\max}$ denote the diameter of the decision space (i.e., the length of the interval). Recall that $C_0(ALG) = \sum_{t=1}^T c^t(\hat{x}^t) + \|\hat{x}^t - \hat{x}^{t-1}\|$ and $OPT_s = \min_x \sum_{t=1}^T c^t(x)$. Then we know that $C_0(ALG) \le C_1(ALG) + D\sum_{t=1}^T \|x^{t+1} - x^t\| + \|d_{\max}\|$ for some constant $D$ by (5.1). Based on our assumptions, we have $\sum_t c^t(\hat{x}^{t+1}) \le OPT_{\|\cdot\|_{ALG}} + O(1)$ and $\sum_t \|\hat{x}^{t+1} - \hat{x}^t\| \le \beta OPT_{\|\cdot\|_{ALG}} + O(1)$. For convenience, we let $E = D + 1 = O(1)$. Then $C_0(ALG)$ is at most:

$$\begin{aligned}
&\sum_{t=1}^T c^t(\hat{x}^{t+1}) + E\|\hat{x}^{t+1} - \hat{x}^t\| + \|d_{\max}\| + O(1)\\
\le&(1 + E\beta)OPT_{\|\cdot\|_{ALG}} + \|d_{\max}\| + O(1)\\
\le&(1 + E\beta)(OPT_s + \|d_{\max}\|_{ALG}) + \|d_{\max}\| + O(1)
\end{aligned}$$

Therefore, we get a regret $C_0(ALG) - OPT_s$ at most

$$\begin{aligned}
&E\beta OPT_s + \|d_{\max}\|(1 + E(1+\beta)\theta) + O(1)\\
=&O(\beta OPT_s + (1+\beta)\theta) = O(\max\{\beta OPT_s, (1+\beta)\theta\})
\end{aligned}$$

In the OCO setting, the cost functions $c^t(x)$ are bounded from below by 0 and are typically bounded from above by a value independent of $T$, e.g., [62, 86], so that $OPT_s = O(T)$. This immediately gives the result that the regret is at most $O(\max\{\beta T, (1+\beta)\theta\})$.

## Appendix 5.E Proof of Lemma 5.4

In this section, we argue that the expected operating cost of RBG (when evaluated under $\|\cdot\|$) with input norm $N(\cdot) = \theta\|\cdot\|$, $\theta \geq 1$, is at most the cost of the optimal dynamic offline algorithm under norm $N$ (i.e., $OPT_N$). Let $M$ denote our decision space. Before proving this result, let us introduce a useful lemma. Let $\hat{x}^1, \hat{x}^2, \ldots, \hat{x}^{T+1}$ denote the actions chosen by RBG (similarly, let $x^1_{OPT}, \ldots, x^{T+1}_{OPT}$ denote the actions chosen by $OPT_N$).

**Lemma 5.6.** $w^t(\hat{x}^{t+1}) = w^{t-1}(\hat{x}^{t+1}) + c^t(\hat{x}^{t+1})$.

*Proof.* We know that for any state $x \in M$, we have $w^t(x) = \min_{y\in M}\{w^{t-1}(y) + c^t(y) + \theta\|x - y\|\}$. Suppose instead $w^t(\hat{x}^{t+1}) = w^{t-1}(y) + c^t(y) + \theta\|\hat{x}^{t+1} - y\|$ for some $y \neq \hat{x}^{t+1}$. Then

$$\begin{aligned} Y^{t+1}(\hat{x}^{t+1}) &= w^t(\hat{x}^{t+1}) + \theta r\|\hat{x}^{t+1}\| \\ &= w^{t-1}(y) + c^t(y) + \theta\|\hat{x}^{t+1} - y\| + \theta r\|\hat{x}^{t+1}\| \\ &> w^{t-1}(y) + c^t(y) + \theta r\|y\| \\ &= Y^{t+1}(y), \end{aligned}$$

which contradicts $\hat{x}^{t+1} = \arg\min_{y\in M} Y^{t+1}(y)$. Therefore $w^t(\hat{x}^{t+1}) = w^{t-1}(\hat{x}^{t+1}) + c^t(\hat{x}^{t+1})$. □

Now let us prove the expected operating cost of RBG is at most the total cost of the optimal solution, $OPT_N$.

$$\begin{aligned} & Y^{t+1}(\hat{x}^{t+1}) - Y^t(\hat{x}^t) \\ \geq & Y^{t+1}(\hat{x}^{t+1}) - Y^t(\hat{x}^{t+1}) \\ = & (w^t(\hat{x}^{t+1}) + \theta r\|\hat{x}^{t+1}\|) - (w^{t-1}(\hat{x}^{t+1}) + \theta r\|\hat{x}^{t+1}\|) \\ = & c^t(\hat{x}^{t+1}) \end{aligned}$$

Lemma 5.4 is proven by summing up the above inequality for $t = 1, \ldots, T$, since $Y^{T+1}(\hat{x}^{T+1}) \leq Y^{T+1}(x^{T+1}_{OPT})$ and $\mathbb{E}Y^{T+1}(x^{T+1}_{OPT}) = OPT_N$ by $\mathbb{E}r = 0$.

Note that this approach also holds when the decision space $F \subset \mathbb{R}^n$ for $n > 1$.

## Appendix 5.F Proof of Lemma 5.5

To prove Lemma 5.5 we make a detour and consider a version of the problem with a discrete state space. We first show that on such spaces the lemma holds for a discretization of RBG, which we name DRBG. Next, we show that as the discretization becomes finer, the solution (and hence switching

cost) of DRBG approaches that of RBG. The lemma is then proven by showing that the optimal cost of the discrete approximation approaches the optimal cost of the continuous problem.

To begin, define a discrete variant of SOCO where the number of states is finite as follows. Actions can be chosen from $m$ states, denoted by the set $M = \{x_1, \ldots, x_m\}$, and the distances $\delta = x_{i+1} - x_i$ are the same for all $i$. Without loss of generality we define $x_1 = 0$. Consider the following algorithm.

**Algorithm 5.3** (Discrete RBG, DRBG($N$))**.**
*Given a norm $N$ and discrete states $M = \{x_1, \ldots, x_m\}$, define $w^0(x) = N(x)$ and $w^t(x) = \min_{y \in M}\{w^{t-1}(y) + c^t(y) + N(x - y)\}$ for all $x \in M$. Generate a random number $r \in (-1, 1)$. For each time step $t$, go to the state $x^t$ which minimizes $Y^t(x^t) = w^{t-1}(x^t) + rN(x^t)$.*

Note that DRBG looks nearly identical to RBG except that the states are discrete. DRBG is introduced only for the proof and need never be implemented; thus we do not need to worry about the computational issues when the number of states $m$ becomes large.

## Bounding the switching cost of DRBG

We now argue that the expected switching cost of DRBG (evaluated under the norm $\|\cdot\|$ and run with input norm $N(\cdot) = \theta\|\cdot\|$) is at most the total cost of the optimal solution in the discrete system (under norm $N$). We first prove a couple of useful lemmas. The first lemma states that if the optimal way to get to some state $x$ at time $t$ is to come to state $y$ in the previous time step, incur the operating cost at state $y$, and travel from state $y$ to state $x$, then in fact the optimal way to get to state $y$ at time $t$ is to come to $y$ at the previous time step and incur the operating cost at state $y$.

**Lemma 5.7.** *If $\exists x, y : w^t(x) = w^{t-1}(y) + c^t(y) + \theta\|x - y\|$, then $w^t(y) = w^{t-1}(y) + c^t(y)$.*

*Proof.* Suppose towards a contradiction that $w^t(y) < w^{t-1}(y) + c^t(y)$. Then we have:

$$w^t(y) + \theta\|x - y\| < w^{t-1}(y) + c^t(y) + \theta\|x - y\| = w^t(x) \leq w^t(y) + \theta\|x - y\|$$

(since one way to get to state $x$ at time $t$ is to get to state $y$ at time $t$ and travel from $y$ to $x$). This is a contradiction, which proves the lemma. □

We now prove the main lemma. Let $SC^t = \sum_{i=1}^{t} \|x^i - x^{i-1}\|$ denote the total switching cost incurred by DRBG up until time $t$, and define the potential function $\phi^t = \frac{1}{2\theta}(w^t(x_1) + w^t(x_m)) - \frac{\|x_m - x_1\|}{2}$. Then we can show the following lemma.

**Lemma 5.8.** *For every time step $t$, $\mathbb{E}SC^t \leq \phi^t$.*

*Proof.* We will prove this lemma by induction on $t$. At time $t = 0$, clearly it is true since the left hand side $\mathbb{E}SC^0 = 0$, while the right hand side $\phi^0 = \frac{1}{2\theta}(w^0(x_1) + w^0(x_m)) - \frac{\|x_m - x_1\|}{2} = \frac{1}{2\theta}(0 + \theta\|x_m - x_1\|) - \frac{\|x_m - x_1\|}{2} = 0$. We now argue that at each time step, the increase in the left hand side is at most the increase in the right hand side.

Since the operating cost is convex, it is non-increasing until some point $x_{min}$ and then non-decreasing over the set $M$. We can imagine our cost vector arriving in $\epsilon$-sized increments as follows. We imagine sorting the cost values so that $c^t(i_1) \leq c^t(i_2) \leq \cdots \leq c^t(i_m)$, and then view time step $t$ as a series of smaller time steps where we apply a cost of $\epsilon$ to all states for the first $c^t(i_1)/\epsilon$ time steps, followed by applying a cost of $\epsilon$ to all states except state $i_1$ for the next $c^t(i_2) - c^t(i_1)/\epsilon$ time steps, etc., where $\epsilon$ has a very small value. If adding this epsilon-sized cost vector would cause us to exceed the original cost $c^t(i_k)$ for some $k$, then we just use the residual $\epsilon' < \epsilon$ in the last round in which state $i_k$ has non-zero cost. Eventually, these $\epsilon$-sized cost vectors will add up precisely to the original cost vector $c^t$. Under these new cost vectors, the behavior of our algorithm will not change (and the optimal solution cannot get worse). Moreover, we would never move to a state in which $\epsilon$ cost was added. If the left hand side does not increase at all from time step $t-1$ to $t$, then the lemma holds (since the right hand side can only increase). Our expected switching cost is the probability that the algorithm moves multiplied by the distance moved. Suppose the algorithm is currently in state $x$. Observe that there is only one state the algorithm could be moving from (state $x$) and only one state $y$ the algorithm could be moving to (we can choose $\epsilon$ to be sufficiently small in order to guarantee this). Notice that the algorithm would only move to a state $y$ to which no cost was added. First we consider the case $x \geq x_{min}$.

The only reason we would move from state $x$ is if $w^t(x)$ increases from the previous time step, so that we have $w^t(x) = w^{t-1}(x)+\epsilon$. Notice that for any state $z > x$, we must have $w^t(z) = w^{t-1}(z)+\epsilon$. If not (i.e., $w^t(z) < w^{t-1}(z) + \epsilon$), then we get a contradiction as follows. The optimal way to get to $z$ at time step $t$, $w^t(z)$, must go through some point $j$ in the previous time step and incur the operating cost at $j$. If $j \geq x$, then we know $w^{t-1}(j) + \epsilon + \theta\|z - j\| = w^t(z) < w^{t-1}(z) + \epsilon \leq w^{t-1}(j)+\theta\|z-j\|+\epsilon$, which cannot happen. On the other hand, by Lemma 5.7, if $j < x$ then we get $w^t(x)+\theta\|z-x\| \leq w^t(j)+\theta\|1\||x-j|+\theta\|1\||z-x| = w^t(j)+\theta\|z-j\| = w^{t-1}(j)+c^t(j)+\theta\|z-j\| = w^t(z) < w^{t-1}(z) + \epsilon \leq w^{t-1}(x) + \theta\|z - x\| + \epsilon$, which cannot happen either. Hence, we know $w^t(z) = w^{t-1}(z) + \epsilon$ for all $z \geq x$.

By the above argument, we can conclude a couple of facts. The state $y$ we move to cannot be such that $y \geq x$. Moreover, we also know that $w^t(x_m) = w^{t-1}(x_m) + \epsilon$ (since $x_m \geq x$). Notice that for us to move from state $x$ to state $y$, the random value $r$ must fall within a very specific range. In

particular, we must have $Y^t(x) \le Y^t(y)$ and $Y^{t+1}(y) \le Y^{t+1}(x)$:

$$\begin{aligned}&(w^{t-1}(x) + \theta r\|1\|x \le w^{t-1}(y) + \theta r\|1\|y) \ \wedge \ (w^t(y) + \theta r\|1\|y \le w^t(x) + \theta r\|1\|x)\\ \Longrightarrow &w^{t-1}(y) - w^{t-1}(x) - \epsilon \le w^t(y) - w^t(x) \le \theta r\|x-y\| \le w^{t-1}(y) - w^{t-1}(x)\end{aligned}$$

This means $r$ must fall within an interval of length at most $\epsilon/\theta\|x-y\|$. Since $r$ is chosen from an interval of length 2, this happens with probability at most $\epsilon/(2\theta\|x-y\|)$. Hence, the increase in our expected switching cost is at most $\|x-y\| \cdot \epsilon/(2\theta\|x-y\|) = \epsilon/2\theta$. On the other hand, the increase in the right hand side is $\phi^t - \phi^{t-1} = \frac{1}{2\theta}(w^t(x_1) - w^{t-1}(x_1) + w^t(x_m) - w^{t-1}(x_m)) \ge \epsilon/2\theta$ (since $w^t(x_m) = w^{t-1}(x_m) + \epsilon$). The case when $x < x_{min}$ is symmetric. This finishes the inductive claim. □

Now we prove the expected switching cost of DRBG is at most the total cost of the optimal solution for the discrete problem.

By Lemma 5.8, for all times $t$ we have $\mathbb{E}SC^t \le \phi^t$. Denote by $OPT^t$ the optimal solution at time $t$ (so that $OPT^t = \min_x w^t(x)$ and $OPT^T = OPT_N$). Let $x^* = argmin_x w^t(x)$ be the final state which realizes $OPT^t$ at time $t$. We have, for all times $t$:

$$\begin{aligned}\mathbb{E}SC^t \le &\phi^t = \frac{1}{2\theta}(w^t(x_1) + w^t(x_m)) - \frac{\|x_m - x_1\|}{2}\\ \le &\frac{1}{2\theta}(w^t(x^*) + \theta\|x^* - x_1\| + w^t(x^*) + \theta\|x_m - x^*\|) - \frac{\|x_m - x_1\|}{2}\\ = &\frac{1}{\theta}OPT^t.\end{aligned}$$

In particular, the equation holds at time $T$, which gives the bound.

## Convergence of DRBG to RBG

In this section, we are going to show that if we keep splitting $\delta$, the output of DRBG, which is denoted by $x_D^t$, converges to the output of RBG, which is denoted by $x_C^t$.

**Lemma 5.9.** *Consider a SOCO with $F = [x_L, x_H]$. Consider a sequence of discrete systems such that the state spacing $\delta \to 0$ and for each system, $[x_1, x_m] = F$. Let $x_i$ denote the output of DRBG in the $i$th discrete system, and $\hat{x}$ denote the output of RBG in the continuous system. Then the sequence $\{x_i\}$ converges to $\hat{x}$ with probability 1 as $i$ increases, i.e., for all $t$, $\lim_{i\to\infty} |x_i^t - \hat{x}^t| = 0$ with probability 1.*

*Proof.* To prove the lemma, we just need to show that $x_i$ converges pointwise to $\hat{x}$ with probability 1. For a given $\delta$, let $Y_D^t$ denote the $Y^t$ used by DRBG in the discrete system (with feasible set $M = \{x_1, \dots, x_m\} \subset F$) and $Y_C^t$ denote the $Y^t$ used by RBG in the continuous system (with

feasible set $F$). The output of DRBG and RBG at time $t$ are denoted by $x_D^t$ and $x_C^t$ respectively. The subsequence on which $|x_C^t - x_D^t| \leq 2\delta$ clearly has $x_D^t$ converge to $x_C^t$. Now consider the subsequence on which this does not hold. For each such system, we can find a $\bar{x}_C^t \in \{x_1, \ldots, x_m\}$ and $|\bar{x}_C^t - x_C^t| < \delta$ (and thus $|\bar{x}_C^t - x_D^t| \geq \delta$) such that $Y_C^t(\bar{x}_C^t) \leq Y_C^t(x_D^t)$, by the convexity[5] of $Y_C^t$. Moreover $Y_D^t(x_D^t) \leq Y_D^t(\bar{x}_C^t)$ and $Y_C^t(x_D^t) \leq Y_D^t(x_D^t)$. So far, we have only rounded the $t$th component. Now let us consider a scheme that will round to the set $M$ all components $\tau < t$ of a solution to the continuous problem.

For an arbitrary trajectory $x = (x^t)_{t=1}^T$, define a sequence $x_R(x)$ with $x_R^\tau \in \{x_1, \ldots, x_m\}$ as follows. Let $l = \max\{k : x_k \leq x^\tau\}$. Set $x_R^\tau$ to $x_l$ if $c^\tau(x_l) \leq c^\tau(x_{l+1})$ or $l = m$, and $x_{l+1}$ otherwise. This rounding will increase the switching cost by at most $2\theta\|\delta\|$ for each timeslot. If $l = m$ then the operating cost is unchanged. Next, we bound the increase in operating cost when $l < m$.

For each timeslot $\tau$, depending on the shape of $c^\tau$ on $(x_l, x_{l+1})$, we may have two cases: (i) $c^\tau$ is monotone; (ii) $c^\tau$ is non-monotone. In case (i), the rounding does not make the operating cost increase for this timeslot. Note that if $x_C^\tau \in \{x_L, x_H\}$ then for all sufficiently small $\delta$, case (ii) cannot occur, since the location of the minimum of $c^\tau$ is independent of $\delta$. We now consider case (ii) with $x_C^\tau \in (x_L, x_H)$. Note that there must be a finite left and right derivative of $c^\tau$ at all points in $(x_L, x_H)$ for $c^\tau$ to be finite on $F$. Hence these derivatives must be bounded on any compact subset of $(x_L, x_H)$. Since $x_C^\tau \in (x_L, x_H)$, there exist a set $[x'_L, x'_H] \subset (x_L, x_H)$ independent of $\delta$ such that for sufficiently small $\delta$ we have $[x_l, x_{l+1}] \subset [x'_L, x'_H]$. Hence there exists a $H^\tau$ such that for sufficiently small $\delta$ the gradient of $c^\tau$ is bounded by $H^\tau$ on $[x_l, x_{l+1}]$. Thus, for sufficiently small $\delta$, the rounding will make the operating cost increase by at most $H^\tau\delta$ in timeslot $\tau$.

Define $H = \max_\tau\{H^\tau\}$. If we apply this scheme to the trajectory which achieves $Y_C^t(\bar{x}_C^t)$, we get a decision sequence in the discrete system with $cost + r\theta\|\bar{x}_C^t\|$ not more than $Y_C^t(\bar{x}_C^t) + (H\delta + 2\theta\|\delta\|)t$ (by the foregoing bound on the increase in costs) and not less than $Y_D^t(\bar{x}_C^t)$ (because the solution of $Y_D^t(\bar{x}_C^t)$ minimizes $cost + r\theta\|\bar{x}_C^t\|$). Specifically, we have $Y_D^t(\bar{x}_C^t) \leq Y_C^t(\bar{x}_C^t) + (H\delta + 2\theta\|\delta\|)t$. Therefore,

$$Y_C^t(\bar{x}_C^t) \leq Y_C^t(x_i^t) = Y_C^t(x_D^t) \leq Y_D^t(x_D^t) \leq Y_D^t(\bar{x}_C^t) \leq Y_C^t(\bar{x}_C^t) + (H\delta + 2\theta\|\delta\|)t.$$

Notice that the gradient bound $H$ is independent of $\delta$ and so $(H\delta + 2\theta\|\delta\|)t \to 0$ as $\delta \to 0$. Therefore, $|Y_C^t(x_i^t) - Y_C^t(\bar{x}_C^t)|$ converges to 0 as $i$ increases.

Independent of the random choice $r$, the domain of $w_C^t(\cdot)$ can be divided into countably many non-singleton intervals on which $w_C^t(\cdot)$ is affine, joined by intervals on which it is strictly convex. Then $Y_C^t(\cdot)$ has a unique minimum unless $-r$ is equal to the slope of one of the former intervals, since $Y_C^t(\cdot)$ is convex. Hence it has a unique minimum with probability one with respect to the

[5]The minimum of a convex function over a convex set is convex, thus by definition, $w^t$ is a convex function by induction. Therefore, $Y_C^t$ is convex as well.

choice of $r$.

Hence w.p.1, $x_C^t$ is the unique minimum of $Y_C^t$. To see that $Y_C^t(\cdot)$ is continuous at any point $a$, apply the squeeze principle to the inequality $w_C^t(a) \leq w_C^t(x) + \theta\|x - a\| \leq w_C^t(a) + 2\theta\|x - a\|$, and note that $Y_C^t(\cdot)$ is $w^t(\cdot)$ plus a continuous function. The convergence of $|\bar{x}_C^t - x_C^t|$ then implies $|Y_C^t(\bar{x}_C^t) - Y_C^t(x_C^t)| \to 0$ and thus $|Y_C^t(x_i^t) - Y_C^t(x_C^t)| \to 0$, or equivalently $Y_C^t(x_i^t) \to Y_C^t(x_C^t)$. Note that the restriction of $Y_C^t$ to $[x_L, x_C^t]$ has a well-defined inverse $Y^{-1}$, which is continuous at $Y_C^t(x_C^t)$. Hence for the subsequence of $x_i^t$ such that $x_i^t \leq x_C^t$, we have $x_i^t = Y^{-1}(Y_C^t(x_i^t)) \to Y^{-1}(Y_C^t(x_C^t)) = x_C^t$. Similarly, the subsequence such that $x_i^t \geq x_C^t$ also converge to $x_C^t$. □

## Convergence of OPT in discrete system

To show that the competitive ratio hold for RBG, we also need to show that the optimal costs converge to those of the continuous system.

**Lemma 5.10.** *Consider a SOCO problem with $F = [x_L, x_H]$. Consider a sequence of discrete systems such that the state spacing $\delta \to 0$ and for each system, $[x_1, x_m] = F$. Let $OPT_D^i$ denote the optimal cost in the $i$th discrete system, and $OPT_C$ denote the optimal cost in the continuous system (both under the norm $N$). Then the sequence $\{OPT_D^i\}$ converges to $OPT_C$ as $i$ increases, i.e., $\lim_{i\to\infty} |OPT_D^i - OPT_C| = 0$.*

*Proof.* We can apply the same rounding scheme in the proof of Lemma 5.9 to the solution vector of $OPT_C$ to get a discrete output with total cost bounded by $OPT_C + (H\delta + 2\theta\|\delta\|)T$, thus

$$OPT_D^i \leq OPT_C + (H\delta + 2\theta\|\delta\|)T.$$

Notice that the gradient bound $H$ is independent of $\delta$ and so $(H\delta + 2\theta\|\delta\|)T \to 0$ as $\delta \to 0$. Therefore, $OPT_D^i$ converges to $OPT_C$ as $i$ increases. □

# Bibliography

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

[2] J. Abernethy, P. L. Bartlett, N. Buchbinder, and I. Stanton. A regularization approach to metrical task systems. In *Proc. Algorithmic Learning Theory (ALT)*, pages 270–284, 2010.

[3] S. Albers. Energy-efficient algorithms. *Comm. of the ACM*, 53(5):86–96, 2010.

[4] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. In *Lecture Notes in Computer Science (STACS)*, volume 3884, pages 621–633, 2006.

[5] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan. Robust and flexible power-proportional storage. In *Proc. ACM SoCC*, 2010.

[6] L. L. H. Andrew, S. Barman, K. Ligett, M. Lin, A. Meyerson, A. Roytman, and A. Wierman. A tale of two metrics: Simultaneous bounds on competitiveness and regret. In *Proc. Conf. on Learning Theory (COLT)*, 2013.

[7] L. L. H. Andrew, M. Lin, and A. Wierman. Optimality, fairness and robustness in speed scaling designs. In *Proc. ACM SIGMETRICS*, 2010.

[8] C. L. Archer and M. Z. Jacobson. Supplying baseload power and reducing transmission requirements by interconnecting wind farms. *J. Appl. Meteorol. Climatol.*, 46:1701–1717, Nov. 2007.

[9] M. Asawa and D. Teneketzis. Multi-armed bandits with switching penalties. *IEEE Trans. Automatic Control*, 41(3):328 –348, Mar. 1996.

[10] J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. *SIAM Journal on Computing*, 37(5):1499–1516, 2008.

[11] B. Avi-Itzhak, H. Levy, and D. Raz. A resource allocation fairness measure: properties and bounds. *Queueing Systems Theory and Applications*, 56(2):65–71, 2007.

[12] N. Bansal, H.-L. Chan, T.-W. Lam, and L.-K. Lee. Scheduling for speed bounded processors. In *Proc. Int. Colloq. Automata, Languages and Programming*, pages 409–420, 2008.

[13] N. Bansal, H.-L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. In *Proc. ACM-SIAM Symp. Discrete Algorithms (SODA)*, pages 693–701, 2009.

[14] N. Bansal, H.-L. Chan, K. Pruhs, and D. Katz. Improved bounds for speed scaling in devices obeying the cube-root rule. In *Automata, Languages and Programming*, pages 144–155, 2009.

[15] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow times. In *Proc. ACM-SIAM SODA*, pages 805–813, 2007.

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

[17] L. A. Barroso and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 4(1):1–108, 2009.

[18] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. Technical Report CLOUDS-TR-2010-3, Univ. of Melbourne, 2010.

[19] M. Bienkowski. Price fluctuations: To buy or to rent. In *Approximation and Online Algorithms*, pages 25–36, 2008.

[20] N. Bingham, C. Goldie, and J. Teugels. *Regular Variation*. Cambridge University Press, 1987.

[21] D. L. Black and D. D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Carnegie Mellon University, 1989.

[22] A. Blum and C. Burch. On-line learning and the metrical task system problem. *Machine Learning*, 39(1):35–58, 2000.

[23] A. Blum, C. Burch, and A. Kalai. Finely-competitive paging. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 450–457, 1999.

[24] A. Blum, S. Chawla, and A. Kalai. Static optimality and dynamic search-optimality in lists and trees. In *Proc ACM-SIAM Symp. Discrete Algorithms (SODA)*, pages 1–8, 2002.

[25] A. Blum, H. Karloff, Y. Rabani, and M. Saks. A decomposition theorem and bounds for randomized server problems. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 197–207, 1992.

[26] A. Blum and Y. Mansour. From external to internal regret. *Learning Theory*, LNCS 3559:621–636, 2005.

[27] P. Bodik, M. P. Armbrust, K. Canini, A. Fox, M. Jordan, and D. A. Patterson. A case for adaptive datacenters to conserve energy and improve reliability. Technical Report UCB/EECS-2008-127, University of California at Berkeley, 2008.

[28] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[29] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[30] J. R. Bradley. Optimal control of a dual service rate M/M/1 production-inventory model. *European Journal of Operations Research*, 161(3):812–837, 2005.

[31] N. Buchbinder, S. Chen, J. Naor, and O. Shamir. Unified algorithms for online learning and competitive analysis. In *Proc. Conf. on Learning Theory (COLT)*, 2012.

[32] D. P. Bunde. Power-aware scheduling for makespan and flow. *J. Scheduling*, 12(5), Oct. 2009.

[33] G. C. Calafiore. Multi-period portfolio optimization with linear control policies. *Automatica*, 44(10):2463–2473, 2008.

[34] H.-L. Chan, J. Edmonds, T.-W. Lam, L.-K. Lee, A. Marchetti-Spaccamela, and K. Pruhs. Nonclairvoyant speed scaling for flow and energy. In *Proc. STACS*, pages 255–264, 2009.

[35] H.-L. Chan, J. Edmonds, and K. Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In *Proc. ACM Symp. Parallel Algorithms and Architectures (SPAA)*, pages 1–10, 2009.

[36] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proc. Symp. Operating System Principles*, pages 103–116, 2001.

[37] 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.

[38] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *Proc. ACM SIGMETRICS*, 2005.

[39] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. USENIX NSDI*, pages 273–286, 2005.

[40] A. Coté, A. Meyerson, and L. Poplawski. Randomized k-server on hierarchical binary trees. In *Proc. ACM Symposium on the Theory of Computing (STOC)*, 2008.

[41] T. M. Cover. Universal portfolios. *Mathematical Finance*, 1(1):1–29, 1991.

[42] T. B. Crabill. Optimal control of a service facility with variable exponential service times and constant arrival rate. *Management Science*, 18(9):560–566, 1972.

[43] R. Das, J. O. Kephart, C. Lefurgy, G. Tesauro, D. W. Levine, and H. Chan. Autonomic multi-agent management of power and performance in data centers. In *Proceedings of International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008.

[44] J. Edmonds. Scheduling in the dark. In *Proc. ACM STOC*, pages 179–188, 1999.

[45] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proc. ACM Int. Symp. Comp. Arch.*, 2007.

[46] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, Nov. 2010.

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

[48] J. M. George and J. M. Harrison. Dynamic control of a queue with adjustable service rate. *Oper. Res.*, 49(5):720–731, 2001.

[49] D. Gmach, Y. Chen, A. Shah, J. Rolia, C. Bash, T. Christian, and R. Sharma. Profiling sustainability of data centers. In *Proc. ISSST*, 2010.

[50] D. Gmach, J. Rolia, C. Bash, Y. Chen, T. Christian, and A. Shah. Capacity planning and power management to exploit sustainable energy. In *Proc. of CNSM*, 2010.

[51] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Proc. IEEE Symp. Workload Characterization (IISWC)*, pages 171–180, Boston, MA, Sept. 2007.

[52] S. Greenberg, E. Mills, B. Tschudi, P. Rumsey, and B. Myatt. Best practices for data centers: Lessons learned from benchmarking 22 data centers. *Proceedings of the ACEEE Summer Study on Energy Efficiency in Buildings in Asilomar, CA. ACEEE, August*, 3:76–87, 2006.

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

[54] S. Guha and K. Munagala. Multi-armed bandits with metric switching costs. In *Automata, Languages and Programming*, volume 5556 of *Lecture Notes in Computer Science*, pages 496–507. Springer Berlin / Heidelberg, 2009.

[55] Y. Gur, O. Besbes, and A. Zeevi. Non-stationary online stochastic approximation. In *Presented at INFORMS general meeting*, 2012.

[56] J. Hamilton. The cost of latency. `http://perspectives.mvdirona.com/`, 2009.

[57] J. Hamilton. Cost of power in large-scale data centers. `http://perspectives.mvdirona.com/`, 2009.

[58] S. V. Hanly. Congestion measures in DS-CDMA networks. *IEEE Trans. Commun.*, 47(3):426–437, Mar. 1999.

[59] M. Harchol-Balter, K. Sigman, and A. Wierman. Asymptotic convergence of scheduling policies with respect to slowdown. *Perf. Eval.*, 49(1-4):241–256, Sept. 2002.

[60] E. Hazan, A. Agarwal, and S. Kale. Logarithmic regret algorithms for online convex optimization. *Mach. Learn.*, 69:169–192, Dec. 2007.

[61] E. Hazan and C. Seshadhri. Efficient learning algorithms for changing environments. In *Proc. International Conference on Machine Learning*, pages 393–400. ACM, 2009.

[62] M. Herbster and M. K. Warmuth. Tracking the best expert. *Mach. Learn.*, 32(2):151–178, Aug. 1998.

[63] T. Horvath and K. Skadron. Multi-mode energy management for multi-tier server clusters. In *Proc. ACM PACT*, page 1, 2008.

[64] S. Irani, R. Gupta, and S. Shukla. Competitive analysis of dynamic power management strategies for systems with multiple power savings states. In *Proc. Design, Automation, and Test in Europe*, page 117, 2002.

[65] S. Irani, S. Shukla, and R. Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embed. Comput. Syst.*, 2(3):325–346, 2003.

[66] V. Joseph and G. de Veciana. Variability aware network utility maximization. *ArXiv Computing Research Repository (CoRR)*, abs/1111.3728, 2011.

[67] V. Joseph and G. de Veciana. Jointly optimizing multi-user rate adaptation for video transport over wireless systems: Mean-fairness-variability tradeoffs. In *INFOCOM, 2012 Proceedings IEEE*, pages 567–575. IEEE, 2012.

[68] A. Kansal, J. Liu, A. Singh, , R. Nathuji, and T. Abdelzaher. Semantic-less coordination of power management and application performance. In *ACM SIGOPS*, pages 66–70, 2010.

[69] S. Kaplan. Power plants: Characteristics and costs. *Congressional Research Service*, 2008.

[70] A. R. Karlin, C. Kenyon, and D. Randall. Dynamic tcp acknowledgement and other stories about $e/(e-1)$. In *Proc. ACM Symp. Theory of Computing (STOC)*, 2001.

[71] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):77–119, 1988.

[72] F. P. Kelly. *Reversibility and Stochastic Networks*. Wiley, 1979.

[73] B. Khargharia, S. Hariri, and M. Yousif. Autonomic power and performance management for computing systems. *Cluster computing*, 11(2):167–181, Dec. 2007.

[74] A. A. Kherani and R. Nunez-Queija. TCP as an implementation of age-based scheduling: fairness and performance. In *IEEE INFOCOM*, 2006.

[75] L. Kleinrock. *Queueing Systems Volume II: Computer Applications*. Wiley Interscience, 1976.

[76] J. Koomey. Growth in data center electricity use 2005 to 2010. In *Analytics Press*, Oakland, CA, USA, 2011.

[77] E. Koutsoupias and C. H. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000.

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

[79] W. Kwon and A. Pearson. A modified quadratic cost problem and feedback stabilization of a linear system. *IEEE Trans. Automatic Control*, AC-22(5):838–842, 1977.

[80] W. H. Kwon, A. M. Bruckstein, and T. Kailath. Stabilizing state feedback design via the moving horizon method. *Int. J. Contr.*, 37(3):631–643, 1983.

[81] T.-W. Lam, L.-K. Lee, I. K. K. To, and P. W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *Proc. Euro. Symp. Alg.*, 2008.

[82] M. Lin, Z. Liu, A. Wierman, and L. L. H. Andrew. Online algorithms for geographical load balancing. In *International Green Computing Conference(IGCC)*, 2012.

[83] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. In *Proc. IEEE INFOCOM*, 2011.

[84] M. Lin, A. Wierman, A. Roytman, A. Meyerson, and L. L. Andrew. Online optimization with switching cost. *ACM SIGMETRICS Performance Evaluation Review*, 40(3):98–100, 2012.

[85] M. Lin, A. Wierman, and B. Zwart. Heavy-traffic analysis of mean response time under shortest remaining processing time. In *Performance Evaluation vol. 68(10)*, Oct. 2011.

[86] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, Feb. 1994.

[87] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. H. Andrew. Geographic load balancing with renewables. In *Proc. ACM GreenMetrics*, San Jose, CA, 7 Jun 2011.

[88] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. H. Andrew. Greening geographical load balancing. In *Proc. ACM SIGMETRICS*, pages 233–244, San Jose, CA, 7-11 Jun 2011.

[89] M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for on-line problems. In *Proc. ACM Symp. Theory of Computing (STOC)*, pages 322–333, 1988.

[90] D. Q. Mayne and H. Michalska. Receding horizon control of nonlinear systems. *IEEE Trans. Automat. Contr.*, 35(7):814–824, 1990.

[91] C. Miller. Solar-powered data centers. *Datacenter Knowledge*, 13 July 2010.

[92] R. Miller. Google data center FAQ. *Datacenter Knowledge*, 27 March 2008.

[93] R. Miller. Facebook installs solar panels at new data center. *Datacenter Knowledge*, 16 April 2011.

[94] D. Mills. Advances in solar thermal electricity technology. *Solar Energy*, 76:19–31, 2004.

[95] L. Minas and B. Ellison. *Energy efficiency for information technology: How to reduce power consumption in servers and data centers*. Intel Press, 2009.

[96] R. Motwani, S. Phillips, and E. Torng. Nonclairvoyant scheduling. *Theoret. Comput. Sci.*, 130(1):17–47, 1994.

[97] S. Ong, P. Denholm, and E. Doris. The impacts of commercial electric utility rate structure elements on the economics of photovoltaic systems. Technical Report NREL/TP-6A2-46782, National Renewable Energy Laboratory, 2010.

[98] E. Pakbaznia and M. Pedram. Minimizing data center cooling and server power costs. In *Proc. ISLPED*, 2009.

[99] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load balancing and unbalancing for power and performacne in cluster-based systems. In *Proc. Compilers and Operating Systems for Low Power*. 2001.

[100] C. G. Plaxton, Y. Sun, M. Tiwari, , and H. Vin. Reconfigurable resource scheduling. In *ACM SPAA*, 2006.

[101] K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. *ACM Trans. Algorithms*, 4(3):Article 38, June 2008.

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

[103] I. A. Rai, G. Urvoy-Keller, and E. Biersack. Analysis of FB scheduling for job size distributions with high variance. In *Proc. ACM SIGMETRICS*, 2003.

[104] L. Rao, X. Liu, L. Xie, and W. Liu. Minimizing electricity cost: Optimization of distributed Internet data centers in a multi-electricity-market environment. In *IEEE INFOCOM*, 2010.

[105] W. Sandmann. A discrimination frequency based queueing fairness measure with regard to job seniority and service requirement. In *Proc. Euro NGI Conf. on Next Generation Int. Nets*, 2005.

[106] R. Stanojevic and R. Shorten. Distributed dynamic speed scaling. In *IEEE INFOCOM*, 2010.

[107] R. E. Tarjan. Amortized computational complexity. *SIAM J. Alg. Disc. Meth.*, 6(2):306–318, 1985.

[108] E. Thereska, A. Donnelly, and D. Narayanan. Sierra: a power-proportional, distributed storage system. Technical Report MSR-TR-2009-153, Microsoft Research, 2009.

[109] P. Tsiaflakis, Y. Yi, M. Chiang, and M. Moonen. Fair greening for DSL broadband access. In *GreenMetrics*, 2009.

[110] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely. Dynamic resource allocation and power management in virtualized data centers. In *Proc. IEEE NOMS*, Apr. 2010.

[111] `http://rredc.nrel.gov/solar/new_data/confrrm/`. 2010.

[112] `http://wind.nrel.gov`. 2010.

[113] `http://www.eia.doe.gov`.

[114] `http://www.google.com/green/operations/renewable-energy.html`.

[115] H. N. Van, F. Tran, and J.-M. Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Computer and Information Technology, 2009. CIT '09. Ninth IEEE International Conference on*, OCT 2009.

[116] X. Wang and M. Chen. Cluster-level feedback power control for performance optimization. In *IEEE HPCA*, pages 101–110, 2008.

[117] R. R. Weber and S. Stidham. Optimal control of service rates in networks of queues. *Adv. Appl. Prob.*, 19:202–218, 1987.

[118] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. Donar: decentralized server selection for cloud services. In *Proc. ACM SIGCOMM*, pages 231–242, New York, NY, USA, 2010. ACM.

[119] A. Wierman. Fairness and classifications. *Perf. Eval. Rev.*, 34(4):4–12, 2007.

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

[121] A. Wierman, L. L. H. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems: Optimality and robustness. *In preparation*, 2010.

[122] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to unfairness in an M/GI/1. In *Proc. ACM SIGMETRICS*, 2003.

[123] W. Xu, X. Zhu, S. Singhal, and Z. Wang. Predictive control for dynamic resource allocation in enterprise data centers. In *Proc. IEEE/IFIP Netw. Op. Manag. Symp (NOMS)*, pages 115–126, 2006.

[124] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 374–382, 1995.

[125] S. Zhang and K. S. Catha. Approximation algorithm for the temperature-aware scheduling problem. In *Proc. IEEE Int. Conf. Comp. Aided Design*, pages 281–288, Nov. 2007.

[126] Y. Zhang, M. Murata, H. Takagi, and Y. Ji. Traffic-based reconfiguration for logical topologies in large-scale WDM optical networks. *IEEE J. Lightwave Technology*, 23(10):2854, 2005.

[127] M. Zink, O. Künzel, J. Schmitt, and R. Steinmetz. Subjective impression of variations in layer encoded videos. In *Proceedings of the 11th International Conference on Quality of Service*, pages 137–154. Springer-Verlag, 2003.

[128] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In T. Fawcett and N. Mishra, editors, *Proc. Int. Conf. Machine Learning (ICML)*, pages 928–936. AAAI Press, 2003.