Algorithmic Issues in Green Data Centers

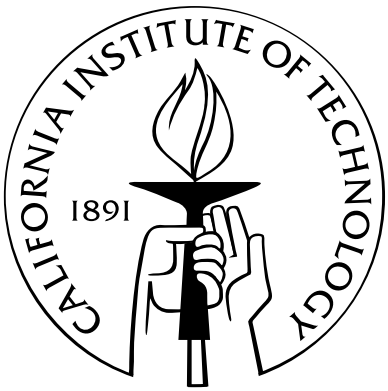
Thesis by

Minghong Lin

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science



California Institute of Technology

Pasadena, California

2011

(Submitted November 1, 2010)

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 and Zhenhua Liu. It is a great pleasure to work with them. They have always provided insightful discussions and constructive suggestions.

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 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 career in research.

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

Power consumption imposes a significant cost for data centers. Thus, it is not surprising that optimizing energy cost in data center is receiving increasing attention. In this thesis, we focus on the algorithmic issues at three levels of energy optimization for data centers: server level, local data center level and global data center level. At the server level, we analyze the common speed scaling algorithms in both worst-case model and stochastic model to answer some fundamental issues in the design of speed scaling algorithms. At the local data center level, we develop an online algorithm to make data center more power-proportional by dynamically adapting the number of active servers to match the current workload. At the global data center level, we propose a framework to explore the diversity of power prices and the diversity of propagation delays given geographically distributed data centers.

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, 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% and is expected to almost double by 2011 if current trends continue. 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.

The natural way to reduce energy costs is to improve energy efficiency, so that the same amount of computational work can be done with less energy. Energy efficiency is broadly defined as the amount of computational work performed divided by the total energy used in the process. 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 is greater than 2 for typical data centers [13].

Fortunately, PUE can be substantially improved by careful design for energy efficiency. The most energy efficient data centers today have PUE ≈ 1.2 [13]. This is achieved via maintaining data centers at a higher temperature to save energy associated with the cost of increasing the equipment failure rate, using more efficient air-flow control to reduce the energy needed for cooling, adopting more efficient gear to reduce the UPS and power distribution losses, etc.

Beyond these engineering improvements, there is also significant energy reduction to be achieved via improved IT design. The energy efficiency of IT components has been widely studied from both a systems perspective, which is well surveyed in [15], and an algorithmic perspective, which can be found in the review article [2]. The literature 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 issues*** at all three levels in this thesis.

Energy efficiency at server level

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 offline solution, and randomized algorithms can do even better [26]. Many generalizations of this problem have been studied, including stochastic settings [7].

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. [43] in 1995. Since [43], three main performance objectives balancing energy and delay have been considered: (i) minimize the total energy used in order to meet job deadlines [11, 34], (ii) minimize the average response time given an energy/power budget [18, 45], and (iii) minimize a linear combination of expected response time and energy usage per job [3, 10].

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 details of all these results about speed scaling are presented in Chapter 2 and in [6, 5, 41, 32].

Energy efficiency at local data center level

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" [14], i.e., use power only in proportion to the load. However, even a typical energy-efficient server still consumes about half its full power when doing virtually no work [14]. A promising approach for making data centers more power-proportional is 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 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 [21], network state [20] and storage state [37, 4]. 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 offline 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 is 3-competitive. Third, 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 details are described in Chapter 3 and in [31].

Energy efficiency at global data center level

The algorithmic questions at this 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, [35] 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 [40] 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 are developing 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. Given the energy optimization at the data center level, our goal is to find a global dispatching scheme to route the workload from different regions to certain data centers dynamically. Depending on how fast the prices and the workloads are changing, we may need epoch-by-epoch centralized optimizations or fully dynamic control-based distributed algorithms. Details of our proposed approach are in Chapter 4.

Chapter 2

Optimization at the server level

Computer systems must make a fundamental tradeoff between performance and energy usage. Speed scaling designs adapt the speed of the system so as to balance these metrics. 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 to balance energy and performance, except when idle (*gated-static speed scaling*). The analytic study of the speed scaling problem began with Yao et al. [43] in 1995. In this work, we focus on minimizing a linear combination of expected response time and energy usage per job [3, 10]. 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).

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., [34], 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. In this work, we focus on the design of speed scalers for both SRPT and PS. For background on SRPT and PS, see [44, 42].

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, denoted $\mathcal{E}$:

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

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

$$\lambda z = \mathbb{E}[N] + \mathbb{E}[P]/\beta \tag{2.2}$$

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 $\mathcal{E}(t) = \int_0^t P(s(t))dt$.

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 [41]. 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 π that defines the order in which jobs are processed, and a speed scaling rule Σ that defines the speed as a function of system state, in terms of the power function, P. 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.[1]

The scheduling algorithms π we consider are online, and so are not aware of a job j until it arrives at time $r(j)$, at which point π 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; 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.

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

In the worst-case model we consider finite, arbitrary (maybe adversarial) instances of arriving jobs. A problem instance consists of ν jobs, with the jth job having arrival time (release time) $r(j)$

[1] Note that for some other objectives, it is better to base the speed on the unfinished work instead [12].

and size (work) x_j. 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. Given an instance I, denote the cost of a given algorithm $\mathcal{A}$ as $z^{\mathcal{A}}(I)$ and the cost of the optimal offline algorithm as $z^{O}(I)$.We study the competitive ratio, defined as

$$CR = \sup_{I} z^{\mathcal{A}}(I)/z^{O}(I).$$

In the stochastic model, we consider an M/GI/1 queue with arrival rate λ. Let X denote a random job size with c.d.f. $F(x)$ and $\rho = \lambda\mathbb{E}[X] \in [0, \infty)$ denote the load of arriving jobs. When the power function is $P(s) = s^{\alpha}$, it is natural to use a scaled load, $\gamma := \rho/\beta^{\alpha}$, which jointly characterizes the impact of ρ and β (see [41]). 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, ρ. 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-cast analysis

There has been significant work studying speed scaling in the worst-case model following Yao et al.'s seminal 1995 paper [43], most of it focusing on SRPT. A promising algorithm that has emerged is (SRPT, $P^{-1}(n)$), and there has been a significant stream of papers providing upper bounds on the competitive ratio of this algorithm for objective (2.1): for unit-size jobs in [3, 12] and for general jobs with $P(s) = s^{\alpha}$ in [9, 30]. A major breakthrough was made in [10], 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. Formally, we have the following theorem, which is proven in Appendix A.

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

Theorem 1 can easily be extended to non-negative power functions by applying the same argument as used in [10]:

Corollary 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.*

Second, we prove that no "natural" speed scaling algorithm can be better than 2-competitive. Roughly, "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 the formal definition in [6]). 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 worst-case analysis of speed scaling under PS. We prove that (PS, $P^{-1}(n\beta)$) is $O(1)$-competitive, and in particular is $(4\alpha - 2)$-competitive for typical α, i.e., $\alpha \in (1, 3]$. Formally, the theorem can be stated as follows and the proof can be found in Appendix B.

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

This builds on [19], which studies LAPS, another policy "blind" to job sizes. (LAPS, $P^{-1}(n\beta)$) is also $O(1)$-competitive for $P(s) = s^\alpha$ with fixed α. However, for both PS and LAPS the competitive ratio is unbounded for large α, which [19] proves holds for all blind policies. But, note that $\alpha \in (1, 3]$ in most computer systems today (e.g., disks, chips, and servers); thus, asymptotics in α are less important than the performance for small α.

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., [14]. 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 ν jobs, PS is $\Omega(\nu^{1/3})$-competitive for mean response time in the constant speed model [33], but is $O(1)$-competitive in the speed scaling model. Again, we conjecture that this holds more generally than for just PS.

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 later, 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 [17, 22, 24, 39][2]. 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 bounds for the optimal speeds have been characterized in [41]. Interestingly, the bounds are tight for large n and have a form similar to the form of the worst-case speeds for SRPT and PS in Theorems 1 and 2.

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 approach. Note that the speeds suggested by the worst-case results for SRPT and PS (Theorems 1 and 2) are the same, *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 lower bound. that was proven in [41] 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 [24, 41], and then these speeds are also used for SRPT. We describe the results from only one of many settings we investigated.

Figure 2.1 shows how the total cost (2.1) depends on the choice of speeds and scheduler, where PS-INV is for (PS, $P^{-1}(n\beta)$), SRPT-INV is for (SRPT, $P^{-1}(n\beta)$), PS-DP is for PS with the optimal speeds of $M/GI/1$ PS, and SRPT-DP is for SRPT with the optimal speeds of $M/GI/1$

[2]This work actually studies the M/M/1 FCFS queue, but since the M/GI/1 PS queue with controllable service rates is a symmetric discipline [27] it has the same occupancy distribution and mean delay as an M/M/1 FCFS queue.

PS. The job size distribution is Pareto(2.2) and the power function is $P(s) = s^2$. 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. Finally, the SRPT-DP scheme performs nearly optimally, which justifies the heuristic of using the optimal speeds for PS in the case of SRPT. 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 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.

To derive the optimal speed s_{gs}, note that, since the power cost is constant at $P(s_{gs})$ whenever the server is running, the optimal speed is

$$s_{gs} = \arg\min_s \beta \mathbb{E}[T] + \frac{1}{\lambda} P(s) \Pr(N \neq 0) \tag{2.3}$$

In the second term $\Pr(N \neq 0) = \rho/s$, setting the derivative to 0 gives that the optimal gated-static speed. The optimal gated-static speed under PS is given in [41]. Unfortunately, things are not as easy in the case of SRPT. The complexity of $\mathbb{E}[T]$ for SRPT rules out calculating the speeds analytically. So, instead we use simpler forms for $\mathbb{E}[T]$ that are exact in asymptotically heavy-traffic [32] and use them to derive the gated-static speed. Details are shown in [6].

2.4 Optimality, robustness and fairness

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 2.2 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. Thus, the simplest policy can nearly match the performance of the most sophisticated policy. Moreover, there is

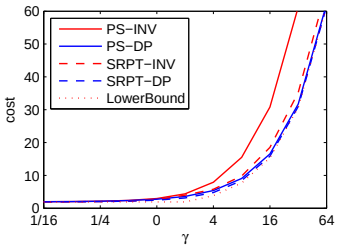


Figure 2.1: Comparison of SRPT and PS under both $s_n = P^{-1}(n\beta)$ and speeds optimized for an M/GI/1 PS system, using Pareto(2.2) job sizes and $P(s) = s^2$.

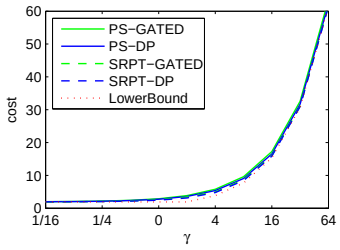


Figure 2.2: Comparison of SRPT and PS under gated-static speeds and dynamic speeds optimized for an M/GI/1 PS, using Pareto(2.2) job sizes and $P(s) = s^2$.

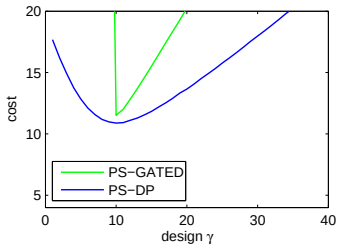


Figure 2.3: Impact of misestimate of γ under PS: cost when $\gamma = 10$, but s_n were chosen for "designed γ", using Pareto(2.2) job sizes and $P(s) = s^2$.

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. This reinforces what we observed for dynamic speed scaling. In addition to observing numerically that the gated-static schemes are near optimal, there is some analytic support for this fact as well. The following corollary has been proven in [6]:

Corollary 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 ρ.*

We just showed 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 ρ. 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, ρ and F, and to model inaccuracies.

Figure 2.3 illustrates that if a gated-static design is used, performance degrades dramatically when ρ is mispredicted for PS (SRPT is similar). 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 2.3 illustrates simulation experiments which show that dynamic speed scaling (PS-DP) is significantly more robust to misprediction of the workload. In fact, we can prove this analytically by providing worst-case guarantees. Note that the corollary below is distinctive in that it provides worst-case guarantees for a stochastic control policy. We omit the proof of the corollary, which can be found in [6].

Corollary 3. *Consider $P(s) = s^\alpha$ with $\alpha \in (1, 2]$ and algorithm $\mathcal{A}$ which chooses speeds s_n^{DP} optimal for PS scheduling in an M/GI/1 queue with load ρ. If $\mathcal{A}$ uses either PS or SRPT scheduling, then*

$\mathcal{A}$ is $O(1)$-competitive in the worst-case model.

To this point we have seen that speed scaling has many benefits; however we can show 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 [38]. 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.

It can be proved that the service-rate differential can lead to unfairness in a rigorous sense under SRPT and non-preemptive policies (e.g. FCFS). However, under PS, speed scaling does not lead to unfairness. Note that gated-static scaling does not magnify unfairness, regardless of the scheduling discipline, since all jobs are processed at the same speed. For the fairness issues of speed scaling, please check [6] for details.

Chapter 3

Optimization 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 [14]. 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.

To answer this question, we develop a general model that captures the major issues that affect the design of a right-sizing algorithm and analytically characterize the optimal offline solution. Further, we introduce and analyze a novel, practical online algorithm and prove that this algorithm is 3-competitive. Finally, we validate our algorithm using two load traces to evaluate the cost savings achieved via dynamic right-sizing in practice.

3.1 Model and notation

We now describe the model we use to explore the cost savings possible via dynamic right-sizing of data centers. The assumptions used in the model are minimal and capture many properties of current data centers and traces we have obtained.

3.1.1 The workload model

We 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 $t \in \{0, 1, \ldots, T\}$. The mean arrival rate for slot t is denoted by λ_t. For convenience, we enforce that $\lambda_t = 0$ for all $t \leq 0$ and all $t \geq T$. We assume that the job interarrival times are much shorter than the timeslot,

so that provisioning can be based on the average arrival rate during a slot. In practice, T could be a year and a slot t could be 10 minutes.

The analytic results of Sections 3.2 and 3.3 assume that the workload has a finite duration, i.e. $T < \infty$, but make no other assumptions about λ_t, i.e., λ_t can be arbitrary. Thus, the analytic results provide worst-case guarantees. However, to provide realistic cost estimates, we consider case-studies in Section 3.4 where λ_t is defined using real-world traces.

3.1.2 The data center cost model

We model a data center as a collection of homogeneous servers.[1] We focus on two important decisions for the data center: (i) determining x_t, the number of active servers during each time slot t, and (ii) assigning arriving jobs to servers, i.e., determining $\lambda_{i,t}$, the arrival rate to server i at time t. (Note that $\sum_{i=1}^{x_t} \lambda_{i,t} = \lambda_t$.) The data center wants to choose x_t and $\lambda_{i,t}$ to minimize the cost during $[1, T]$. Our model for the cost focuses on the server costs of the data center.[2]

We model the cost of a server by (i) the operating costs incurred by an active server and (ii) the switching costs to toggle a server into and out of a power-saving mode (e.g., off/on or sleeping/waking). Both components include energy and delay costs.

The *operating costs* are modeled by a convex function $f(\lambda_{i,t})$, which is the same for all servers. The convexity assumption is quite general and captures many common server models. One example of a convex cost model is a weighted sum of delay costs and energy costs: $r(\lambda_{i,t}, d) + e(\lambda_{i,t})$, where $r(\lambda_{i,t}, d)$ is the revenue lost given delay d and arrival rate $\lambda_{i,t}$, and $e(\lambda_{i,t})$ is the energy cost of an active server handling arrival rate $\lambda_{i,t}$. One common model of the energy cost for typical servers is an affine function $e(\lambda_{i,t}) = e_0 + e_1\lambda_{i,t}$ where e_0 and e_1 are constants; e.g., see [1]. The lost revenue is more difficult to model. One natural model for it is $r(\lambda_{i,t}, d) = d_1\lambda_{i,t}(d - d_0)^+$ 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. Further, the average delay can be modeled using standard queuing theory results. For example, if the server happens to be modeled by an M/GI/1 Processor Sharing queue then $d = 1/(1 - \lambda_{i,t})$, where the service rate of the server is assumed to be 1 without loss of generality [28]. The combination of these models gives

$$f(\lambda_{i,t}) = d_1\lambda_{i,t}\left(\frac{1}{1-\lambda_{i,t}} - d_0\right)^+ + (e_0 + e_1\lambda_{i,t}) \tag{3.1}$$

The above is one example that convex $f(\cdot)$ can capture, but the results hold for any convex model of operating costs. 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

[1] Multiple classes of servers may be incorporated at the cost of added notational complexity.
[2] Minimizing server energy consumption also reduces cooling and power distribution costs [14].

the 99th percentile is $\log(100)/(1-\lambda)$, and so the form of (3.1) does not change [28]. Similarly, when servers use dynamic speed scaling, if the energy cost is modeled as polynomial in speed as in [43], then the aggregate cost $f(\cdot)$ remains convex [41, 6]. Note that, in practice, $f(\cdot)$ can be empirically measured by observing the system over time.

The *switching cost*, β, models the cost of toggling a server back-and-forth between active and power-saving modes. The constant β includes the costs of (i) the energy used toggling a server, (ii) the delay in migrating connections/data/etc. (e.g., via VM techniques) 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 β is on the order of the cost to run a server for a few seconds (waking from suspend-to-RAM) or migrating network state [20] or storage state [37], to several minutes (to migrate a large VM [21]). However, if (iii) is included, then β becomes on the order of the cost to run a server for an hour [16]. 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 β may be many hours' server costs.

Note that this model ignores many issues surrounding reliability and availability, which are key components of data center service level agreements (SLAs). In practice, a solution that toggles servers must still maintain the reliability and availability guarantees. For example, if data is replicated three times and two copies fail while the third is asleep, the third copy must immediately be woken. Modeling such failures is beyond the scope of this work, however previous work shows that solutions are possible [37].

3.1.3 The data center optimization problem

Given the cost models above, the goal of the data center is to choose the number of active servers x_t and the dispatching rule $\lambda_{i,t}$ to minimize the total cost during $[1, T]$, which is captured by the following optimization:

$$\begin{aligned} \text{minimize} \quad & \sum_{t=1}^{T}\sum_{i=1}^{x_t} f(\lambda_{i,t}) + \beta\sum_{t=1}^{T}(x_t - x_{t-1})^+ \\ \text{subject to} \quad & 0 \le \lambda_{i,t} \le 1 \text{ and } \sum_{i=1}^{x_t}\lambda_{i,t} = \lambda_t, \end{aligned} \tag{3.2}$$

where the constraint $\lambda_{i,t} \le 1$ is a result of normalizing the arrival rate, without loss of generality, such that an arrival rate of 1 is the largest that a server can stabilize. Note that we model the cost β of toggling a server as being incurred when the server is returned to an active state. Though the data center seeks to solve (3.2), it must do so in an *online* manner, i.e, at time τ, it does not have full information about λ_t for $t > \tau$.

In the remainder of this section we simplify the form of (3.2) by noting that, if x_t is fixed, then

the remaining optimization for $\lambda_{i,t}$ is convex. Thus, we can use the KKT conditions to determine the optimal dispatching rule $\lambda^*_{i,t}$. This yields that $\lambda^*_{1t} = \lambda^*_{2t} = \cdots = \lambda_t/x_t$, which implies that once x_t is fixed the optimal dispatching rule is to "load balance" across the servers. Given that load balancing is always optimal, we can decouple dispatching ($\lambda_{i,t}$) from capacity planning (x_t), and simplify (3.2) into purely a capacity planning optimization:

$$\begin{aligned} \text{minimize} \quad & \sum_{t=1}^{T} x_t f(\lambda_t/x_t) + \beta \sum_{t=1}^{T} (x_t - x_{t-1})^+ \\ \text{subject to} \quad & x_t \geq \lambda_t \end{aligned} \tag{3.3}$$

It is this formulation of the data center optimization that we focus on for the remainder of the work. Note that $x_t f(\lambda_t/x_t)$ is the perspective function of the convex function $f(\cdot)$, thus it is also convex. Therefore, (3.3) is a convex optimization problem for x_t. Throughout, denote the operating cost of a vector $X = (x_1, \ldots, x_T)$ by $cost_o(X) = \sum_{t=1}^{T} x_t f(\lambda_t/x_t)$, $cost_s(X) = \beta \sum_{t=1}^{T} (x_t - x_{t-1})^+$, and $cost(X) = cost_o(X) + cost_s(X)$.

Formulation (3.3) makes two important simplifications. First, 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. Second, it does not enforce an upper bound on the number of servers active at time t. However, we can include this constraint by simply redefining the operating cost at time t to be infinity if x_t exceeds certain value. The numerical results show that the optimal solution with the additional constraint $x_t < K$ is simply the minimum of K and the solution to (3.3).

3.2 Optimal structure

Given the data center optimization problem, the first natural task is to characterize the optimal offline solution, i.e., the optimal solution given access to the full vector of λ_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 to the data center optimization problem, X^* in terms of two bounds on the optimal solution which correspond to charging β cost either when a server goes into power-saving mode or when comes out. The optimal x^*_τ can be viewed as 'lazily' staying within these bounds going backwards in time.

More formally, let us first describe upper and lower bounds on x^*_τ, denoted x^U_τ and x^L_τ, respectively. Let $(x^L_{\tau,1}, \ldots, x^L_{\tau,\tau})$ be the solution vector to the following optimization problem

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

Then, define $x_\tau^L = x_{\tau,\tau}^L$. Similarly, let $(x_{\tau,1}^U, \dots, x_{\tau,\tau}^U)$ be the solution vector to the following optimization problem

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

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 arrival information for $t > \tau$. In the case of the lower bound, β cost is incurred for each server toggled on, while in the upper bound, β cost is incurred for each server toggled into power-saving mode. Define $(x)_a^b = \max(\min(x, b), a)$ as the projection of x into $[a, b]$. Then, we have:

Theorem 3. *The optimal solution $X^* = (x_0^*, \dots, x_T^*)$ of the data center optimization problem (3.3) satisfies the following backward recurrence relation*

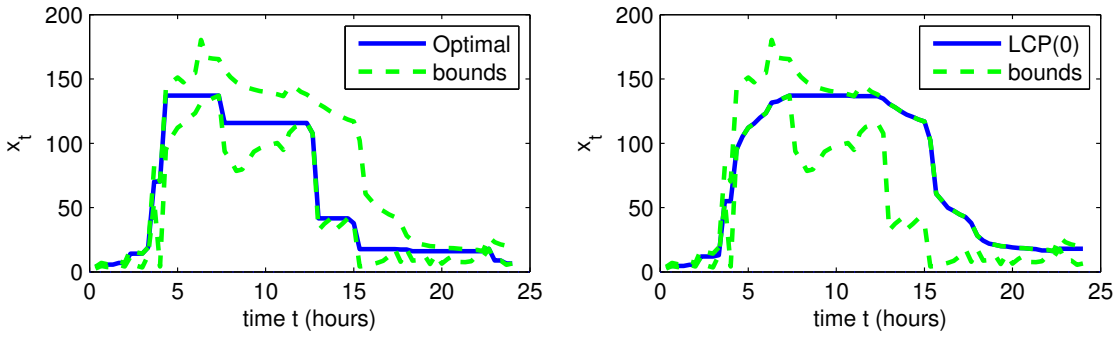
$$x_\tau^* = \begin{cases} 0, & \tau \geq T; \\ (x_{\tau+1}^*)_{x_\tau^L}^{x_\tau^U}, & \tau \leq T-1. \end{cases} \tag{3.6}$$

The proof of Theorem 3 is shown in Appendix C. An example of the optimal x_t^* can be seen in Figure 3.1(a). More numeric examples of the performance of the optimal offline algorithm are provided in Section 3.4.

Theorem 3 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. An important point highlighted by this interpretation is that it is impossible for an online algorithm to compute x_τ^* since, without knowledge of the future, an online algorithm cannot know whether to keep x_τ constant or to follow the upper/lower bound.

3.3 Online algorithm

A major contribution of our work is the presentation and analysis of a novel *online* algorithm, Lazy Capacity Provisioning (LCP(w)). At time τ, LCP(w) knows only λ_t for $t \leq \tau + w$, for some prediction window w. Here, we assume that these are known perfectly, but we show in Section 3.4 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 3.2. 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.



(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 trace with a sampling period of 20 minutes.

Before defining LCP(w) formally, recall that the bounds x_τ^U and x_τ^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 (3.5) and $x_\tau^{L,w} = x_{\tau+w,\tau}^L$ in that of (3.4). Note that $x_\tau^{U,0} = x_\tau^U$ and $x_\tau^{L,0} = x_\tau^L$.

Algorithm 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

$$x_\tau^{LCP(w)} = \begin{cases} 0, & \tau \leq 0; \\ (x_{\tau-1}^{LCP(w)})_{x_\tau^{L,w}}^{x_\tau^{U,w}}, & \tau \geq 1. \end{cases} \tag{3.7}$$

The following theorem provides the worst-cast performance guarantee for LCP(w), which is proven in Appendix D.

Theorem 4. *LCP(w) is 3-competitive for optimization (3.3). Further, for any finite w and $\epsilon > 0$ there exists an instance such that LCP(w) attains a cost greater than $3 - \epsilon$ times the optimal cost.*

Note that Theorem 4 says that the competitive ratio is independent of any parameters of the model, e.g., the prediction window size w, the switching cost β, and the form of the operating cost function $f(\lambda)$. Surprisingly, this means that even the "myopic" LCP(0) is 3-competitive, regardless of the arrival vector, despite having no information about arrivals 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. However, for any fixed w, there is a worst-case arrival sequence and cost function such that the competitive ratio is arbitrarily close to 3.

Finally, though 3-competitive may seem like a large gap, we can further show that the cost of the online algorithm is less than the optimal solution plus two times the switching cost of the optimal

solution, which 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.

3.4 Case study

In this section our goal is two-fold: First, we seek to evaluate the cost incurred by LCP(w) relative to the optimal solution in the context of realistic workloads. Second, more generally, we seek to illustrate the cost savings that come from dynamic right-sizing in data centers. To accomplish these goals, we experiment use real-world traces.

3.4.1 Experimental setup

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

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 very 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.

We consider the cost function (3.1) with $d_0 = 1.5$, $d_1 = 1$, $e_0 = 1$, $e_1 = 0$ and $\beta = 6$. A wide range of settings to investigate the impact of energy cost and switching cost are studied in [31].

The workloads for the experiments are drawn from real-world data center traces from Hotmail and MSR Cambridge. We present the results for MSR trace only. The results for Hotmail trace are similar and can be found in [31]. The trace period was 1 week starting from 5PM GMT on the 22nd February 2007. Loads were averaged over disjoint 10 minute intervals. It has strong diurnal properties and has peak-to-mean ratio (PMRs) of 4.64. A wide range of setting to investigate the impact of PMRs is studied in [31].

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. Given that prediction errors for real data sets tend to be small [25, 29], to simplify our experiments we allow LCP(w) perfect predictions. The impact of prediction error is also studied in [31].

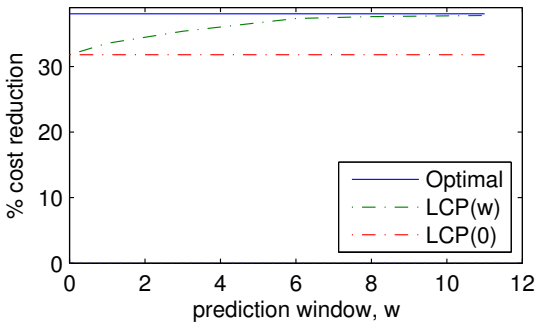


Figure 3.2: Cost saving from dynamic right-sizing for MSR trace.

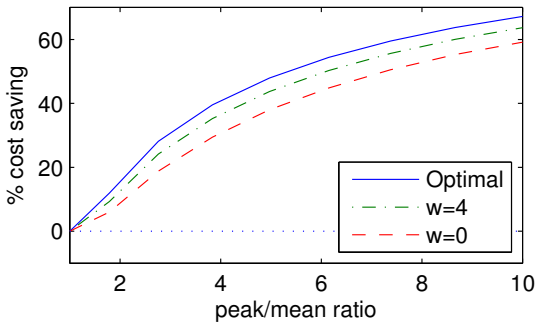


Figure 3.3: Impact of the peak-to-mean ratio (PMR) of the workload for MSR trace.

3.4.2 Is right-sizing beneficial?

Remember that we have attempted to choose experimental settings so that the benefit of dynamic right-sizing is conservatively estimated. The goal is to evaluate the cost incurred by LCP(w) relative to the optimal solution in the context of realistic workloads and illustrate the cost savings that come from dynamic right-sizing in data centers. 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 3.2 illustrates the cost savings of LCP(w), where the unit of w is one timeslot which is 10 minutes.

The first observation from Figure 3.2 is that the savings possible from dynamic right-sizing are significant, where a significant 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.

Dynamic right-sizing inherently exploits the gap between the peaks and valleys of the workload, and intuitively provides larger savings as that gap grows. Figure 3.3 illustrates that this intuition holds. 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. The workload for the figure is generated from the MSR workload by scaling λ_t as $\hat{\lambda}_t = k(\lambda_t)^\alpha$, varying α and adjusting k to keep the mean constant. Note that though Figure 3.3 includes only the results for MSR trace, the resulting plot for the Hotmail trace is nearly identical. This highlights that the cost saving depends primarily on the PMR of the workload.

These results together with the results for a wide range of settings studying the impact of prediction error, energy costs and switching costs in [31] suggest that significant savings are possible

in practice and that savings become dramatic when the workload is predictable over an interval proportional to the toggling cost. The magnitude of the potential savings depend primarily on the peak-to-mean ratio of the workload, with a PMR of 5 being enough to give 40% cost saving even for quite bursty workloads. Further, we find that dynamic right-sizing provides more than 15% cost savings even when the background work available for valley filling makes up $\approx 40\%$ of the workload when the PMR is larger than 3.

Chapter 4

Optimization 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 good 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. However, since the energy cost is becoming a big fraction for the total cost of the data centers, the routing scheme also needs to take into account the power prices at the different data centers. The routing scheme becomes even more challenging when time-varying workloads and time-varying electricity prices are considered. The goal of our proposed work is to provide a distributed algorithm to achieve near optimal routing when considering all these factors.

4.1 Initial model

Our first step is to consider the optimization problem with workloads and electricity prices fixed. This model applies for the scenario when the workloads and the electricity prices change slowly over time so that we can solve the optimization with fixed workloads and electricity prices epoch-by-epoch, or for the scenario when we do not want to change the capacity provisioning and dispatching quickly and thus use the "average" workloads and electricity prices in the optimization.

Assume that there are N data centers, with electricity price p_i and M_i servers at data center i. Assume that each active server consumes a certain amount of energy each time unit, which does not depend on its load, and that an inactive server does not consume any energy. Thus, the energy cost for data center i is proportional to its number of active servers. The mean delay for data center i is $D_i(m_i, \lambda_i)$ with m_i active servers and λ_i workload. There are J regions of users with aggregate traffic L_j generated from region j (e.g., aggregate traffic from each state of US). The round-trip propagation delay from region j to data center i is d_{ij}. Our goal is to minimize a linear combination

of energy cost and user delay. Let γ be the weight to tradeoff energy cost and user delay. We are going to decide the number of active servers m_i for data center i and the traffic λ_{ij} from region j to data centers i. This optimization problem can be formalized as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{N} m_i p_i + \gamma \sum_{j=1}^{J} \sum_{i=1}^{N} \lambda_{ij} (D_i(m_i, \sum_{j=1}^{J} \lambda_{ij}) + d_{ij}) \\ \text{subject to} \quad & \sum_{i=1}^{N} \lambda_{ij} = L_j, \ \lambda_{ij} \geq 0, \ m_i \leq M_i \end{aligned}$$

This formulation does not enforce m_i to be integer. This is acceptable since the number of servers in a typical data center is large. We can see that this formulation is a convex optimization if the function $D_i(m_i, \lambda_i)$ is convex. Actually $D_i(m_i, \lambda_i)$ can be convex under certain queueing assumption about the data centers, e.g., assume each server in data center i is an $M/M/1$ queueing system with arrival rate λ_i/m_i. It becomes much more difficult to handle if $D_i(m_i, \lambda_i)$ is not convex.

4.2 Proposed work

The are many questions to ask based on the above model. Some examples are:

Time-varying workloads and electricity prices

As shown in Chapter 3, the time-varying workloads, especially the diurnal pattern gives us an opportunity to dynamically right-size data center and save lots of energy. Dynamic right-sizing can be explored even further at the global data center level since we have the flexibility of dispatching the traffic across data centers dynamically. However, dynamic right-sizing data center will make the global dispatching more complicated. The dispatching algorithm and the right-sizing algorithm have impact on each other and it seems not easy to decouple. Besides the workload, the electricity prices may also be time-varying, e.g., California electricity prices are dynamic. The electricity prices are expected to become even more dynamic over time due to the renewable power supply such as solar and wind. This may make the online right-sizing and dispatching even more challenging.

Distributed algorithms

Since the data centers are dispersed geographically and the workloads and electricity prices are changing over time, a centralized algorithm may have a large communication overhead, especially once the workloads and electricity prices are changing quickly so that the algorithm has to be run again and again. The most common dispatching techniques used for multiple data centers are HTTP ingress proxies, which are adopted by Google and Yahoo, and the authoritative DNS servers, which are adopted by Akamai and most CDNs. These proxies or DNS servers need to know how to direct the requests and adapt to changing conditions. A centralized algorithm incurs

significant overhead because the proxies or DNS servers need to interact with the central controller. Additionally, centralized algorithm introduces a single point of failure as well as making the system less responsive to sudden changes of workloads. Therefore, we prefer distributed algorithms for solving the optimization problem. The communication needed by the algorithm should be minimal, and the algorithm should be responsive to time-varying workloads and electricity prices.

Unsplittable traffic

In the initial model, the traffic from each region may be split across several data centers. Thus the proxies or the DNS servers have to know not only which data centers to route the traffic but also how much traffic for each data center. Further, even if the workloads and electricity prices are fixed, to maintain data consistency additional mechanisms may be needed to make sure that requests from the same user are dispatched to the same data center over time. Since the capacity of a data center is usually much higher than the aggregate traffic from each region, to make implementation simple and reliable, we may prefer to route the aggregate traffic from each region to only one data center. However, this variation will make the algorithmic problem even harder to solve.

Bibliography

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

[2] S. Albers. Energy-efficient algorithms. *Comm. of the ACM*, 53(5):86–96, 2010.

[3] 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.

[4] 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.

[5] L. L. Andrew, M. Lin, A. Tang, and A. Wierman. Speed scaling to trade-off efficiency, simplicity, robustness and fairness. In *IEEE COMSOC MMTC E-Letter*, 2010.

[6] L. L. H. Andrew, M. Lin, and A. Wierman. Optimality, fairness and robustness in speed scaling designs. In *Proc. ACM Sigmetrics*, 2010.

[7] J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. *SIAM Journal on Computing*, 37(5):1499–1516, 2008.

[8] N. Bansal, H.-L. Chan, J. Edmonds, and K. Pruhs. Preprint, 2009. Available ⟨http://www.cs.pitt.edu/∼kirk/postdoc/spaa.pdf⟩.

[9] 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.

[10] N. Bansal, H.-L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. In *Proc. SODA*, 2009.

[11] 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.

[12] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow times. In *Proc. SODA*, pages 805–813, 2007.

[13] L. A. Barroso and U. Holzle. *The Datacenter as a Computer.*

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

[15] 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, 2010.

[16] 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.

[17] 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.

[18] D. P. Bunde. Power-aware scheduling for makespan and flow. In *Proc. ACM Symp. Parallel Alg. and Arch.*, 2006.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] J. Edmonds. Scheduling in the dark. In *Proc. ACM STOC*, pages 179–188, 1999.

[24] J. M. George and J. M. Harrison. Dynamic control of a queue with adjustable service rate. *Oper. Res.*, 49(5):720–731, 2001.

[25] 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*, Boston, MA, Sept. 2007.

[26] 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.

[27] F. P. Kelly. *Reversibility and Stochastic Networks.* 1979.

[28] L. Kleinrock. *Queueing Systems Volume II: Computer Applications.* Wiley Interscience, 1976.

[29] 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.

[30] 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.*, 2009.

[31] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. Under submission.

[32] M. Lin, A. Wierman, and B. Zwart. Heavy-traffic analysis of mean response time under shortest remaining processing time. Preprint, 2009.

[33] R. Motwani, S. Phillips, and E. Torng. Nonclairvoyant scheduling. *Theoret. Comput. Sci.*, 130(1):17–47, 1994.

[34] K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. In *Scandinavian Worksh. Alg. Theory*, 2004.

[35] L. Rao, X. Liu, L. Xie, and W. Liu. Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment. pages 1 –9, mar. 2010.

[36] R. E. Tarjan. Amortized computational complexity. *SIAM J. Alg. Disc. Meth.*, 6(2):306–318, 1985.

[37] E. Thereska, A. Donnelly, and D. Narayanan. Sierra: a power-proportional, distributed storage system. Technical Report MSR-TR-2009-153, Microsoft Research, 2009.

[38] P. Tsiaflakis, Y. Yi, M. Chiang, and M. Moonen. Fair greening for DSL broadband access. In *GreenMetrics*, 2009.

[39] R. R. Weber and S. Stidham. Optimal control of service rates in networks of queues. *Adv. Appl. Prob.*, 19:202–218, 1987.

[40] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. Donar: decentralized server selection for cloud services. In *SIGCOMM '10: Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, pages 231–242, New York, NY, USA, 2010. ACM.

[41] A. Wierman, L. L. H. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems. In *Proc. IEEE INFOCOM*, 2009.

[42] A. Wierman, J. Lafferty, A. Scheller-wolf, and W. Whitt. Scheduling for todays computer systems: Bridging theory and practice. Technical report, 2007.

[43] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. IEEE FOCS*, pages 374–382, 1995.

[44] S. F. Yashkov and A. S. Yashkova. Processor sharing: A survey of the mathematical theory. *Autom. Remote Control*, 68(9):1662–1731, 2007.

[45] 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.

Appendices

Appendix A

Proof of Theorem 1

The proofs of Theorem 1 (and Theorem 2) use a technique termed *amortized local competitive analysis* [23, 36]. The technique works as follows.

To show that an algorithm $\mathcal{A}$ is c-competitive with an optimal algorithm OPT for a performance metric $z = \int z(t)dt$ 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 Φ is not differentiable, it does not increase;

3. *Running condition:* When Φ is differentiable,

$$z^{\mathcal{A}}(t) + \frac{d\Phi}{dt} \leq cz^{O}(t), \tag{A.1}$$

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

Given these conditions, the competitiveness follows from integrating (A.1), which gives

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

Now let us prove the upper bound in Theorem 1 by amortized local competitive analysis. The proof is a refinement of the analysis in [10] that accounts more carefully for some boundary cases. It uses the potential function:

$$\Phi(t) = \int_0^\infty \sum_{i=1}^{n[q;t]} \Delta(i)\, dq \tag{A.2}$$

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 of Theorem 1.

Lemma 1. *Let $\eta \geq 1$ and Φ be given by (A.2) with*

$$\Delta(i) = \frac{1+\eta}{\beta} P'\left(P^{-1}\left(i\beta\right)\right). \tag{A.3}$$

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

$$n^{\mathcal{A}} + P(s^{\mathcal{A}})/\beta + \frac{d\Phi}{dt} \leq (1+\eta)(n^O + P(s^O)/\beta). \tag{A.4}$$

The proof of Lemma 1 uses the following lemmas, which parallel those in [10]. 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. *For any non-decreasing Δ with $\Delta(i) = 0$ for $i \leq 0$, if $n^O < n^A$ then, under SRPT, where Φ is differentiable either*

$$\textit{both} \quad \tfrac{d}{dt}\Phi \leq \Delta(n^A - n^O + 1)(-s^A + s^O) \tag{A.5a}$$

$$\textit{and} \quad n^O \geq 1, \tag{A.5b}$$

$$\textit{or} \quad \tfrac{d}{dt}\Phi \leq \Delta(n^A - n^O)(-s^A + s^O). \tag{A.5c}$$

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 (A.2), 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

$$d\Phi^A = -\Delta(n^A[q^A] - n^O[q^A])s^A\,dt \tag{A.6a}$$

$$d\Phi^O \leq \Delta(n^A[q^O] - (n^O[q^O] - 1))s^O\,dt. \tag{A.6b}$$

The result follows from one of the following cases, divided by dt. The improvement from [10] comes from handling the boundary case $n^O = 0$ more carefully.

$q^A < q^O$ The second term in (A.6a) 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 Δ 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 (A.2) 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 (A.5c).

If $n^O > 1$ then $q^A > q^O$ implies $n^O[q^A] \leq n^O[q^O] - 1$. Since Δ is non-decreasing, (A.6a) 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 (A.6b) becomes $d\Phi^O \leq \Delta(n^A - n^O + 1)s^O\,dt$. This implies (A.5a) and (A.5b).

□

This differs from the corresponding result in [10] in condition (A.5b), which ensures that the argument of Δ in (A.5) is always at most n^A and gives the following.

Lemma 3. *Consider a regular power function, P, and let $\Delta(i)$ be given by (A.3) for $i > 0$. If $n^O < n^A$ and $n^A \leq P(s^A)/\beta$ then (A.5) implies*

$$\frac{d\Phi}{dt} \leq (1+\eta)(P(s^O)/\beta - n^A + n^O). \tag{A.7}$$

Proof. Since P is regular, Δ is non-decreasing. Now, consider two cases.

If (A.5a) and (A.5b) holds, then let $\Psi(s) = P(s)/\beta$ and set $i = n^A - n^O + 1$ in Lemma 4 below to give

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

where the last inequality follows from

$$n^O \geq 1 \quad \Rightarrow \quad s^A \geq P^{-1}(n^A\beta) \geq \Psi^{-1}(n^A - n^O + 1). \tag{A.8}$$

Otherwise (A.5c) holds. Since $s^A \geq P^{-1}(n^A\beta) \geq \Psi^{-1}(n^A - n^O)$, the above manipulations go through again, with $i = n^A - n^O$ in Lemma 4. □

Next, we need the following result, Lemma 3.1 of [10].

Lemma 4. *[10] Let Ψ be a strictly increasing, strictly convex, differentiable function. Let $i, s^A, s^O \geq 0$ be real. Then*

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

We can now prove Lemma 1.

Proof of Lemma 1. When $n^A = 0$, (A.4) holds trivially. Consider now three cases when $n^A \geq 1$:

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 $n^O[q] - n^A[q] \leq 0$ for all times in $[t, t+dt]$. Since $d\Phi^A \leq 0$ on any interval, $d\Phi \leq 0$. Thus (A.4) holds, since $P(s^A)/\beta \leq \eta n^A$.

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 applies. Then by Lemma 3, $d\Phi/dt \leq (1+\eta)(P(s^O)/\beta - n^A + n^O)$, whence

$$\begin{aligned} n^A + \frac{P(s^A)}{\beta} + \frac{d}{dt}\Phi &\leq n^A + \eta n^A + (1+\eta)(\frac{P(s^O)}{\beta} - n^A + n^O) \\ &= (1+\eta)(n^O + P(s^O)/\beta). \end{aligned} \tag{A.9}$$

Finally, if $n^O = n^A$, then either $d\Phi \leq 0$ or (A.5) 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, and (A.5) holds.

Again, if (A.5) holds, then (A.4) holds. If instead $d\Phi \leq 0$, then the left hand side of (A.4) is at most $(1+\eta)n^A$, which is less than the first term on the right hand side. □

Using Lemma 1, we can now prove Theorem 1.

Proof of Theorem 1. To show that the competitive ratio of (SRPT, $P^{-1}(n\beta)$) is at most 2, we show that Φ given by (A.2) and (A.3) is a valid potential function.

The boundary conditions are satisfied since $\Phi = 0$ when there are no jobs in the system. Also, Φ 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 Φ is unchanged. When a job is completed, $n[q]$ is unchanged for all $q > 0$, and so Φ is again unchanged. The running condition is established by Lemma 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 λ, 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 λ, 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 λ becomes large, this tends to 2 since a regular P is unbounded. □

Appendix B

Proof of Theorem 2

Theorem 2 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

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

where $q^\pi(j;t)$ is the remaining work on job j at time t under scheme π, 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 [19] to analyze LAPS. As a result, to prove Theorem 2, we can use the corresponding results in [19] to verify the boundary and jump conditions. All that remains is the running condition, which follows from the technical lemma below.

Lemma 5. *Let Φ be given by (B.1) 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 Φ is differentiable,*

$$n^A + (s^A)^\alpha/\beta + \frac{d\Phi}{dt} \leq c(n^O + (s^O)^\alpha/\beta) \tag{B.2}$$

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

Proof of Lemma 5. First note that if $n^A = 0$ then the LHS of (B.2) is 0, and the inequality holds. Henceforth, consider the case $n^A \geq 1$.

The rate of change of Φ 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 (B.1).

Let $l \geq 0$ be the number of zero terms in the sum (B.1), corresponding to jobs on which PS is leading OPT. The sum in (B.1) 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 Φ due to PS is at least as negative as

$$-\Gamma \sum_{i=1}^{n^A - l} i^{1-1/\alpha} \frac{s^A}{n^A} \leq -\Gamma \int_0^{n^A - l} i^{1-1/\alpha} \frac{s^A}{n^A}\, di \leq -\Gamma \frac{\alpha}{2\alpha - 1} (n^A - l)^{2-1/\alpha} \beta^{1/\alpha} (n^A)^{(1/\alpha)-1} \tag{B.3}$$

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 (B.2), 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

$$0 \leq c(s^O)^\alpha/(\beta n^A) - \Gamma s^O/(n^A)^{1/\alpha} + cl/n^A + \Gamma \frac{\alpha \beta^{1/\alpha}}{2\alpha - 1} (1 - l/n^A)^{2-1/\alpha} - (1+\eta). \tag{B.4}$$

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 [8], 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

$$-\left(1 - \frac{1}{\alpha}\right) \left(\frac{\beta \Gamma^\alpha}{c\alpha}\right)^{1/(\alpha-1)}. \tag{B.5}$$

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 $\Gamma = (1+\eta)(2\alpha-1)/\beta^{1/\alpha}$. $\square$

Appendix C

Proof of Theorem 3

In this section we will prove Theorem 3, which is based on the following lemma:

Lemma 6. *For all τ, $x_\tau^L \leq x_\tau^* \leq x_\tau^U$.*

Before beginning the proofs, let us first rephrase the data center optimization (3.3) again. To do this, without loss of generality, we define the cost function f such that $f(\lambda) = \infty$ for $\lambda < 0$ and $\lambda > 1$. This allows the removal of the constraint in optimization (3.3) that $x_t \geq \lambda_t$ and the rephrasing as:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{t=1}^{T} x_t f(\lambda_t/x_t) + \beta \sum_{t=1}^{T} y_t \\ \text{subject to} & y_t \geq x_t - x_{t-1}, \text{ and } y_t \geq 0. \end{array} \tag{C.1}$$

Next, we want to work with the dual of optimization (C.1). The Lagrangian is

$$\begin{aligned} L(x, y, \mu) &= \sum_{t=1}^{T} x_t f(\lambda_t/x_t) + \beta \sum_{t=1}^{T} y_t + \sum_{t=1}^{T} \mu_t (x_t - x_{t-1} - y_t) \\ &= \sum_{t=1}^{T} (x_t f(\lambda_t/x_t) + (\beta - \mu_t) y_t) + \sum_{t=1}^{T-1} (\mu_t - \mu_{t+1}) x_t + \mu_T x_T - \mu_1 x_0 \end{aligned}$$

and the dual function is

$$g(\mu) = \inf_{x,y \geq 0} L(x, y, \mu)$$

Since we are interested in the maximum of $g(\mu)$ in the dual problem, we need only to consider the

case $\beta - \mu_t \geq 0$. Then the dual function becomes

$$\begin{aligned} g(\mu, \gamma) &= \inf_x \left(\sum_{t=1}^{T} x_t f(\lambda_t / x_t) + \sum_{t=1}^{T-1} (\mu_t - \mu_{t+1}) x_t + \mu_T x_T - \mu_1 x_0 \right) \\ &= -\sum_{t=1}^{T-1} F^*_{\lambda_t}(\mu_{t+1} - \mu_t) - F^*_{\lambda_T}(-\mu_T) - \mu_1 x_0. \end{aligned}$$

where $F^*_{\lambda_t}(z)$ is the conjugate of the function $F_{\lambda_t}(z) = z f(\lambda_t / z)$. Therefore the corresponding dual problem of (C.1)

$$\begin{aligned} \text{maximize} \quad & -\sum_{t=1}^{T-1} F^*_{\lambda_t}(\mu_{t+1} - \mu_t) - F^*_{\lambda_T}(-\mu_T) - \mu_1 x_0 \\ \text{subject to} \quad & 0 \leq \mu_t \leq \beta, \end{aligned} \tag{C.2}$$

where the complementary slackness conditions are

$$\mu_t(x_t - x_{t-1} - y_t) = 0 \tag{C.3}$$

$$(\beta - \mu_t) y_t = 0, \tag{C.4}$$

and the feasibility condition is $y_t = (x_t - x_{t-1})^+$, for $0 \leq \mu_t \leq \beta$.

Using the above, we now observe a relationship between the data center optimization in (C.1) and the upper and lower bounds, i.e., optimizations (3.5) and (3.4). Specifically, if $\mu_{\tau+1} = 0$ in the solution of optimization (C.2), then $\mu_1, \ldots, \mu_\tau$ is the solution to the following problem:

$$\begin{aligned} \text{maximize} \quad & -\sum_{t=1}^{\tau-1} F^*_{\lambda_t}(\mu_{t+1} - \mu_t) - F^*_{\lambda_\tau}(-\mu_\tau) - \mu_1 x_0 \\ \text{subject to} \quad & 0 \leq \mu_t \leq \beta. \end{aligned} \tag{C.5}$$

Thus, the corresponding $x_1, \ldots, x_\tau$ is the solution to optimization (3.4). On the other hand, if $\mu_{\tau+1} = \beta$ in the solution of optimization (C.2), then $\mu_1, \ldots, \mu_\tau$ is the solution to the following problem:

$$\begin{aligned} \text{maximize} \quad & -\sum_{t=1}^{\tau-1} F^*_{\lambda_t}(\mu_{t+1} - \mu_t) - F^*_{\lambda_\tau}(\beta - \mu_\tau) - \mu_1 x_0 \\ \text{subject to} \quad & 0 \leq \mu_t \leq \beta. \end{aligned}$$

Letting $\mu'_t = \beta - \mu_t$, then gives

$$\begin{aligned} \text{maximize} \quad & -\sum_{t=1}^{\tau-1} F^*_{\lambda_t}(\mu'_t - \mu'_{t+1}) - F^*_{\lambda_\tau}(\mu'_\tau) + \mu'_1 x_0 - \beta x_0 \\ \text{subject to} \quad & 0 \le \mu'_t \le \beta. \end{aligned}$$

which is the dual of the optimization (3.5), thus the corresponding $x_1, \ldots, x_\tau$ is the solution to optimization (3.5).

We require two technical lemmas before moving to the proofs of Lemma 6 and Theorem 3.

Lemma 7. *Given $\Lambda = (\lambda_i, \ldots, \lambda_j)$, $x_{i-1} = \hat{x}_{i-1}$ and $x_j \le \hat{x}_j$, Let $X = (x_i, \ldots, x_{j-1})$ be the solution to optimization (3.4) given Λ with constraints fixing the initial and final values as x_{i-1} and x_j respectively. Let $\hat{X} = (\hat{x}_i, \ldots, \hat{x}_{j-1})$ be the solution to optimization (3.4) given Λ with constraints fixing the initial and final values as $\hat{x}_{i-1}$ and $\hat{x}_j$ respectively. Then, there exists an $\hat{X}$ such that $\hat{X} \ge X$.*

Proof. If there is more than one solution for $\hat{X}$, let $\hat{X}$ be the solution with greatest $\hat{x}_{j-1}$. We will first argue that $x_{j-1} \le \hat{x}_{j-1}$. By definition, we have

$$\sum_{k=i}^{j-1} x_k f(\lambda_k/x_k) + \beta \sum_{k=i}^{j} (x_k - x_{k-1})^+ \le \sum_{k=i}^{j-1} \hat{x}_k f(\lambda_k/\hat{x}_k) + \beta \sum_{k=i}^{j-1} (\hat{x}_k - \hat{x}_{k-1})^+ + \beta (x_j - \hat{x}_{j-1})^+$$

$$\sum_{k=i}^{j-1} \hat{x}_k f(\lambda_k/\hat{x}_k) + \beta \sum_{k=i}^{j} (\hat{x}_k - \hat{x}_{k-1})^+ \le \sum_{k=i}^{j-1} x_k f(\lambda_k/x_k) + \beta \sum_{k=i}^{j-1} (x_k - x_{k-1})^+ + \beta (\hat{x}_j - x_{j-1})^+$$

Note that if the second inequality is an equality, then $x_{j-1} \le \hat{x}_{j-1}$. Otherwise, sum the two inequalities, to obtain the strict inequality

$$(x_j - x_{j-1})^+ + (\hat{x}_j - \hat{x}_{j-1})^+ < (\hat{x}_j - x_{j-1})^+ + (x_j - \hat{x}_{j-1})^+$$

Since $x_j \le \hat{x}_j$, we can conclude that $x_{j-1} < \hat{x}_{j-1}$. Therefore, we always have $x_{j-1} \le \hat{x}_{j-1}$.

Next, recursively consider the subproblem given arrival rates $(\lambda_i, \ldots, \lambda_{j-1})$, $x_{i-1} = \hat{x}_{i-1}$ and $x_{j-1} \le \hat{x}_{j-1}$. The same argument as above yields that $x_{j-2} \le \hat{x}_{j-2}$. This continues and we can conclude that $(x_i, \ldots, x_{j-1}) \le (\hat{x}_i, \ldots, \hat{x}_{j-1})$. □

Lemma 8. *Given $\lambda_1, \ldots, \lambda_\tau$ and additional constraints $x_0 = x_S$ and $x_\tau = x_E$, optimization (3.4) and optimization (3.5) yield the same solution vector.*

Proof. The difference between optimization (3.4) and optimization (3.5) is the objective function.

Denote the objective of (3.4) by $g_L(x_0, \ldots, x_\tau)$ and the objective of (3.5) by $g_U(x_0, \ldots, x_\tau)$, then

$$g_L(x_0, \ldots, x_\tau) - g_U(x_0, \ldots, x_\tau) = \sum_{t=1}^{\tau} (x_t - x_{t-1})^+ - \sum_{t=1}^{\tau} (x_{t-1} - x_t)^+ = \sum_{t=1}^{\tau} (x_t - x_{t-1}) = x_E - x_S,$$

which is a constant. Thus optimization (3.4) and optimization (3.5) yield the same solution vector. □

We now complete the proofs of Lemma 6 and Theorem 3.

Proof of Lemma 6. We begin with some definitions. Let $X_\tau^L = (x_{\tau,1}^L, x_{\tau,2}^L, \ldots, x_{\tau,\tau}^L)$ be the solution of optimization (3.4) at time τ and define X_τ^U symmetrically. Additionally, let X_τ^* be the solution to optimization (3.3) with arrival rates $(\lambda_1, \ldots, \lambda_\tau, 0)$ and constraint $x_{\tau+1} = x_{\tau+1}^* \geq 0$.

Now, consider the optimization problem (3.3) with arrival rate $(\lambda_1, \ldots, \lambda_\tau, 0)$ and constraint $x_{\tau+1} = 0$ and denote its solution by $\{x_t'\}$. Since $x_\tau' \geq \lambda_\tau > 0$, the complementary slackness condition gives that $\mu_{\tau+1} = 0$ in its dual problem. Thus, $\mu_1', \ldots, \mu_\tau'$ is the solution to optimization (C.5), and the corresponding $x_1', \ldots, x_\tau'$ is the solution to optimization (3.4), i.e., $x_t' = x_{\tau,t}^L$. Next, we apply Lemma 7, which gives that $x_\tau^L \leq x_\tau^*$, as desired.

Symmetrically to the argument for the lower bound, together with Lemma 8, we can see that X_τ^U is the solution to optimization (3.3) with arrival rates $(\lambda_1, \ldots, \lambda_\tau, 0)$ and constraint $x_{\tau+1} = \infty$. Further, X_τ^* is the solution of optimization (3.3) with arrival rates $(\lambda_1, \ldots, \lambda_\tau, 0)$ and constraint $x_{\tau+1} = x_{\tau+1}^* < \infty$. Again, applying Lemma 7 gives that for all t, $x_{\tau,t}^* \leq x_{\tau,t}^U$, and thus, in particular, we have that $x_\tau^* \leq x_\tau^U$, as desired. □

Proof of Theorem 3. As a result of Lemma 6, we know that $x_\tau^* \in [x_\tau^L, x_\tau^U]$ for all τ. Further, if $x_\tau^* > x_{\tau+1}^*$, by the complementary slackness condition (C.3), we have that $\mu_{\tau+1} = 0$. Thus, in this case, x_τ^* solves optimization (3.4) for the lower bound, i.e., $x_\tau^* = x_\tau^L$. Symmetrically, if $x_\tau^* < x_{\tau+1}^*$, we have that complementary slackness condition (C.4) gives $\mu_{\tau+1} = \beta$ and so x_τ^* solves optimization (3.5) for the upper bound, i.e., $x_\tau^* = x_\tau^U$. Thus, whenever x_τ^* is increasing/decreasing it must be matching the upper/lower bound, respectively. □

Appendix D

Proof of Theorem 4

In this section we will prove Theorem 4. We begin by proving the following lemma which is a generalization of Lemma 6:

Lemma 9. *$x_\tau^L \le x_\tau^{L,w} \le x_\tau^* \le x_\tau^{U,w} \le x_\tau^U$ for all $w \ge 0$.*

Proof of Lemma 9. First, we prove that $x_\tau^{L,w} \le x_\tau^*$. By definition, $x_\tau^{L,w} = x_{\tau+w,\tau}^L$, and so is the solution to optimization (3.4) given $\lambda_1, \dots, \lambda_{\tau+w}, 0, \dots$ and the added (redundant) constraint that $x_{\tau+w+1} = 0$. Further, we can view the optimal x_τ^* as the solution of optimization (3.4) given $\lambda_1, \dots, \lambda_{\tau+w}, 0, \dots$ with constraint $x_{\tau+w+1} = x_{\tau+w+1}^* \ge 0$. From these two representations, we can apply Lemma 7, to conclude that $x_\tau^{L,w} \le x_\tau^*$.

Next, we prove that $x_\tau^L \le x_\tau^{L,w}$. To see this we notice that $x_\tau^{L,w}$ is also the solution to optimization (3.4) given $\lambda_1, \dots, \lambda_\tau, 0, \dots$ and the added (redundant) constraint that $x_{\tau+1} = x_{\tau+w,\tau+1}^L \ge 0$. Then, x_τ^L is the solution to optimization (3.4) given $\lambda_1, \dots, \lambda_\tau, 0, \dots$ with added (redundant) constraint $x_{\tau+1} = 0$. From these two representations, we can again apply Lemma 7, to conclude that $x_\tau^L \le x_\tau^{L,w}$.

Finally, the proof that $x_\tau^* \le x_\tau^{U,w} \le x_\tau^U$ for all $w \ge 0$ is symmetric and so we omit it. □

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

Corollary 4. *The optimal solution of the data center optimization (3.3) satisfies the following backwards recurrence relation*

$$x_\tau^* = \begin{cases} 0, & \tau \ge T; \\ (x_{\tau+1}^*)_{x_\tau^{L,w}}^{x_\tau^{U,w}}, & \tau \le T-1. \end{cases} \tag{D.1}$$

Moving to the proof of Theorem 4, the first step is to use the above lemmas to characterize the relationship between $x_\tau^{LCP(w)}$ and x_τ^*. Note that $x_0^{LCP(w)} = x_0^* = x_T^{LCP(w)} = x_T^* = 0$.

Lemma 10. *Consider timeslots $0 = t_0 < t_1 < \ldots < t_m = T$ such that $x_{t_i}^{LCP(w)} = x_{t_i}^*$. Then, during each segment (t_{i-1}, t_i), either*

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

(ii) $x_t^{LCP(w)} < x_t^*$ *and both* $x_t^{LCP(w)}$ *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 4 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 $x_t^{U,w}$ or $x_t^{L,w}$, we know that any time t_i where $x_{t_i}^{LCP(w)} = x_{t_i}^*$ and $x_{t_i+1}^{LCP(w)} \neq x_{t_i+1}^*$ implies that both $x_{t_i}^{LCP(w)}$ and $x_{t_i}^*$ are equal to either $x_{t_i}^{U,w}$ or $x_{t_i}^{L,w}$.

Now we must consider two cases. First, consider the case that $x_{t_i+1}^{LCP(w)} > x_{t_i+1}^*$. It is easy to see that $x_{t_i+1}^{LCP(w)}$ doesn't match the lower bound since $x_{t_i+1}^*$ is not less than the lower bound. Note, we have that $x_{t_i}^{LCP(w)} \geq x_{t_i+1}^{LCP(w)}$ 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 $x_{t_i}^* = x_{t_i}^{U,w} \geq x_{t_i+1}^{LCP(w)} > x_{t_i+1}^*$. Both $x_{t_i}^{LCP(w)}$ and $x_{t_i}^*$ match the lower bound. Further, the next time when the optimal solution and LCP(w) match, t_{i+1}, is the next time either the number of servers in LCP(w) matches the lower bound $x_t^{L,w}$ or the next time the number of servers in the optimal solution matches the upper bound $x_t^{U,w}$. 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, and we omit the proof of part (ii) because it is symmetric. □

Given Lemma 10, we bound the switching cost of LCP(w).

Lemma 11. $cost_s(X^{LCP(w)}) = cost_s(X^*)$.

Proof. Consider the sequence of times $0 = t_0 < t_1 < \ldots < t_m = T$ such that $x_{t_i}^{LCP(w)} = x_{t_i}^*$ identified in Lemma 10. 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 12. $cost_o(X^{LCP(w)}) \leq cost_o(X^*) + \beta \sum_{t=1}^{T} |x_t^* - x_{t-1}^*|$.

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 10, 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. Define $X_\tau = (x_{\tau,1}, \ldots, x_{\tau,\tau})$ as the solution vector of optimization (3.5) given $\lambda_1, \ldots, \lambda_\tau$ with the additional constraint that the number of servers at time τ match that chosen by LCP(w), i.e., $x_{\tau,t} = x_\tau^{LCP(w)}$. Additionally, define $X'_{\tau+1} = (x'_{\tau+1,1}, \ldots, x'_{\tau+1,\tau+1})$ as the solution vector of optimization (3.5) on $\lambda_1, \ldots, \lambda_\tau, 0$ with the additional constraint that $x_{\tau+1} = x_\tau^{LCP(w)}$. Note that Lemma 8 gives that $X'_{\tau+1}$ is also the solution vector to optimization (3.4) given $\lambda_1, \ldots, \lambda_\tau, 0$ and additional constraint $x_{\tau+1} = x_\tau^{LCP(w)}$. Finally, define the objective value for a vector $y = (y_1, \ldots, y_m)$ as $c(y) = \sum_{t=1}^m y_t f(\lambda_t/y_t) + \beta \sum_{t=1}^m (y_{t-1} - y_t)^+$.

Our goal is to prove that

$$\sum_{t=t_{i-1}+1}^{t_i} x_t^{LCP(w)} f(\lambda_t / x_t^{LCP(w)}) \leq \sum_{t=t_{i-1}+1}^{t_i} x_t^* f(\lambda_t / x_t^*) + \beta |x_{t_{i-1}} - x_{t_i}|. \tag{D.2}$$

To accomplish this, we first argue that $x'_{\tau+1,\tau} = x_\tau^{LCP(w)}$ via a proof by contradiction. Note that if $x'_{\tau+1,\tau} > x_\tau^{LCP(w)} = x'_{\tau+1,\tau+1}$, then the dual condition (C.3) would imply that $\mu_{\tau+1} = 0$ for the optimization (3.4). Thus $x'_{\tau+1,\tau}$ would belong to the optimal solution of (3.4) in $[1, \tau]$ with constraint $\mu_{\tau+1} = 0$. This would imply that $x'_{\tau+1,\tau} = x_\tau^L$, which contradicts the fact that $x'_{\tau+1,\tau} > x_\tau^{LCP(w)} \geq x_\tau^L$. Second, if $x'_{\tau+1,\tau} < x_\tau^{LCP(w)}$, then we can follow a symmetric argument to arrive at a contradiction. Thus $x'_{\tau+1,\tau} = x_\tau^{LCP(w)}$.

Consequently, $x'_{\tau+1,t} = x_{\tau,t}$ for all $t \in [0, \tau]$. Thus

$$c(X'_{\tau+1}) = c(X_\tau) + x_\tau^{LCP(w)} f(0). \tag{D.3}$$

Next, recalling $x_\tau^{LCP(w)}$ is non-increasing in case (i) by Lemma 10, we have $x_{\tau+1,\tau+1} = x_{\tau+1}^{LCP(w)} \leq x_\tau^{LCP(w)} = x'_{\tau+1,\tau+1}$. It then follows from Lemma 7 that $X_{\tau+1} \leq X'_{\tau+1}$ and thus $x_{\tau+1,\tau} \leq x_\tau^{LCP(w)}$. Therefore, we have:

$$\begin{aligned} c(X'_{\tau+1}) &\leq c((x_{\tau+1,1}, \ldots, x_{\tau+1,\tau}, x_\tau^{LCP(w)})) \\ &= c((x_{\tau+1,1}, \ldots, x_{\tau+1,\tau})) + x_\tau^{LCP(w)} f(0). \end{aligned} \tag{D.4}$$

Combining equations (D.3) and (D.4), we obtain

$$c(X_\tau) \leq c((x_{\tau+1,1}, \ldots, x_{\tau+1,\tau}))$$

whence $c(X_\tau) + x_{\tau+1}^{LCP(w)} f(\lambda_{\tau+1} / x_{\tau+1}^{LCP(w)}) \leq c(X_{\tau+1})$. By summing this equality for $\tau \in [t_{i-1}, t_i)$, we have

$$\sum_{t=t_{i-1}+1}^{t_i} x_t^{LCP(w)} f(\lambda_t / x_t^{LCP(w)}) \leq c(X_{t_i}) - c(X_{t_{i-1}}).$$

Expanding out $c(\cdot)$ then gives (D.2), which completes case (i).

We now move to *case (ii)*, i.e., segments where $x_t^{LCP(w)} < x_\tau^*$ for all $t \in (t_{i-1}, t_i)$. A parallel argument gives that (D.2) holds in this case as well.

To complete the proof we combine the results from case (i) and case (ii), summing equation (D.2) over all segments (and the additional times when $x_t^{LCP(w)} = x_t^*$) and applying Lemma 11.

$$\sum_{t=1}^{T} x_t^{LCP(w)} f(\lambda_t / x_t^{LCP(w)}) \leq \sum_{t=1}^{T} x_t^* f(\lambda_t / x_t^*) + \beta |x_t^* - x_{t-1}^*|.$$

□

We can now prove the competitive ratio in Theorem 4.

Lemma 13. *$cost(X^{LCP(w)}) \leq cost(X^*) + 2cost_s(X^*)$. Thus, LCP(w) is 3-competitive for the data center optimization (3.3).*

Proof. Combining Lemma 12 and Lemma 11 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 4 is to prove that the competitive ratio of 3 is tight.

Lemma 14. *Let $\epsilon > 0$. The competitive ratio of LCP(w) is larger than $3 - \epsilon$.*

Proof. The particular instance which corresponds to the worst-case is defined as follows. The cost function is defined as $f(z) = z^m + f_0$ $(0 \leq z \leq 1, m \geq 2)$, $\beta = 0.5$. The arrival rate at time i is $\lambda_i = \delta^{i-1}$ $(1 < \delta < 1.5)$ for $1 \leq i \leq n$, and $\lambda_i = 0$ for $n < i \leq T$, where $n = \log_\delta \frac{1}{\delta - 1}$, $f_0 = \frac{\beta(\delta^m - 1)}{n(\delta^{mn} - 1)}$ and $T > \beta / f_0 + n$.

For the offline optimization, denote the solution by vector x^*, we know that x_i^* is non-decreasing for $i \in [1, n]$ and $x_i^* = 0$ for $i \in [n+1, T]$. If

$$[x f(\lambda_i / x)]' < 0 \ for \ x \in [\lambda_i, x_n^*] \tag{D.5}$$

then $x_i^* = x_n^*$. Assume condition (D.5) holds for all $i \in [1, n]$, then we have $\sum_{i=1}^{n} \frac{\lambda_i^m}{(x_n^*)^{m-1}} + (nf_0 + \beta) x_n^*$ be the minimum value. If the following condition also holds:

$$\lambda_n / x_n^* \in [0, 1] \tag{D.6}$$

Then by the first order condition we get

$$(x_n^*)^{-m} = \frac{(nf_0 + \beta)(\delta^m - 1)}{(m-1)(\delta^{mn} - 1)} \tag{D.7}$$

It is easy to check that our f_0 and β satisfy both (D.5) and (D.6). Thus (D.7) holds. By substituting x_n^* into the objective function, we get the cost for the offline optimal solution:

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

For the online algorithm LCP(0), denote the result by vector $\hat{x}$. we know $\hat{x}_i$ is non-decreasing for $i \in [1, n]$ and $\hat{x}_n = x_n^*$. By the same argument for x_n^*, we have

$$(\hat{x}_\tau)^{-m} = \frac{(\tau f_0 + \beta)(\delta^m - 1)}{(m-1)(\delta^{m\tau} - 1)} \tag{D.8}$$

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

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

Let $\delta \to 1$, then $(\delta^n - 1)/\delta^n \to 1$. By L'Hospital's Law, we get

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

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

For $\tau > n$, by LCP(0), we know that x_τ will stay constant until it hits the upper bound. Assume that $X_\tau = \{x_{\tau,t}\}$ is the solution of optimization (3.5) in $[1, \tau]$ ($\tau > n$). Now we are going to prove that for τ such that $f_0(\tau - n) < \beta$, $x_{\tau,\tau} \geq x_n^*$, 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

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

We can see that $x_{\tau,i} = x_{\tau,n} (i \in [n+1, \tau])$ is the solution, thus $x_{\tau,\tau} \geq x_n^*$. 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 τ so that $(\tau - n) f_0 \to \beta$, and thus

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

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

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

We can choose a large m and small δ to make it arbitrarily close to 3.

We have finished the proof for LCP(0). Now let us consider LCP(w) ($w > 0$). Denote the solution of LCP(w) by $\hat{x}'$. At time τ, we are solving the same optimization problem as LCP(0) but in $[1, \tau + w]$ for $\tau \in [1, n - w]$, thus $\hat{x}'_\tau = \hat{x}_{\tau+w}$ of LCP(0). Thus

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

By pushing $\delta \to 1$, 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 an τ such that $C'_{[n+1,\tau-w]} \to \beta x_n^*$. Therefore, as $\delta \to 1$, we have $C'_{[1,T]}/C_{[1,T]} \to 1$, thus, We can choose a large m and small δ to make the competitive ratio of LCP(w) arbitrarily close to 3 too. □

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

Lemma 15. *If there exists an index $t \in [1, \tau - 1]$ such that $x^U_{\tau,t+1} < x^U_{\tau,t}$ or $x^L_{\tau,t+1} > x^L_{\tau,t}$, then $(x^U_{\tau,1}, \ldots, x^U_{\tau,t}) = (x^L_{\tau,1}, \ldots, x^L_{\tau,t})$, and no matter what the future arrival is, solving the optimization in $[1, \tau']$ for $\tau' > \tau$ is equivalent to solving two optimizations: one over $[1, t]$ with initial condition x_0 and final condition $x^U_{\tau,t}$ and the second over $[t+1, \tau']$ with initial condition $x^U_{\tau,t}$.*

Proof. Consider the case $x^U_{\tau,t+1} < x^U_{\tau,t}$. We know that $(x^U_{\tau,1}, \ldots, x^U_{\tau,t})$ is the solution of (3.5) on $(\lambda_1, \ldots, \lambda_t, 0)$ with constraint $x_{t+1} = x^U_{\tau,t+1}$, thus it is also the solution of (3.4) on $(\lambda_1, \ldots, \lambda_t, 0)$ with constraint $x_{t+1} = x^U_{\tau,t+1}$. Since $x^U_{\tau,t+1} < x^U_{\tau,t}$, we have $\mu_{t+1} = 0$ for the solution of (3.4), which is identical to the solution of the optimization (3.4) in $[1, t]$ without constraint. Therefore, $(x^U_{\tau,1}, \ldots, x^U_{\tau,t}) = (x^L_{\tau,1}, \ldots, x^L_{\tau,t})$. The proof for the case $x^L_{\tau,t+1} > x^L_{\tau,t}$ is symmetric and so we omit it.

Notice that $(x^U_{\tau,1}, \ldots, x^U_{\tau,t})$ is also the solution of (3.5) on $(\lambda_1, \ldots, \lambda_t, 0)$ with constraint $x_{t+1} = \infty$. thus it is also the solution of (3.4) on $(\lambda_1, \ldots, \lambda_t, 0)$ with constraint $x_{t+1} = \infty$. $(x^L_{\tau,1}, \ldots, x^L_{\tau,t})$ is also the solution of (3.4) on $(\lambda_1, \ldots, \lambda_t, 0)$ with constraint $x_{t+1} = 0$. Not matter what the future arrival is, the solution of the optimization in $[1, \tau']$ must have $x_t \in [0, \infty)$. Based on Lemma 7, we know that $x^L_{\tau,t} \leq x_{t+1} \leq x^U_{\tau,t}$. Since $x^L_{\tau,t} = x^U_{\tau,t}$, we have $x_{t+1} = x^U_{\tau,t}$. Thus not matter what the future arrival is, solving the optimization in $[1, \tau'](\tau' > \tau)$ is equivalent to solving two optimizations: one in $[1, t]$ with constraint x_0 and $x^U_{\tau,t}$, the other one in $[t+1, \tau']$ with constraint $x^U_{\tau,t}$. $\square$