Dynamic right-sizing for power-proportional data centers

Minghong Lin and Adam Wierman
California Institute of Technology

Lachlan L. H. Andrew
Swinburne University of Technology

Eno Thereska
Microsoft Research

***Abstract*—Power consumption imposes a significant cost for data centers implementing cloud services, yet much of that power is used to maintain excess service capacity during periods of predictably low load. This paper investigates how much can be saved by dynamically 'right-sizing' the data center by turning off servers during such periods, and how to achieve that saving via an online algorithm. We prove that the optimal offline algorithm for dynamic right-sizing has a simple structure when viewed in reverse time, and this structure is exploited to develop a new 'lazy' online algorithm, which is proven to be 3-competitive. We validate the algorithm using traces from two real data center workloads and show that significant cost-savings are possible.**

I. Introduction

Energy costs represent a significant fraction of a data center's budget [1] and this fraction is expected to grow as the price of energy increases in coming years. Hence, there is a growing push to improve the energy efficiency of the data centers behind cloud computing. A guiding focus for research into 'green' data centers is the goal of designing data centers that are 'power-proportional', i.e., use power only in proportion to the load. However, current data centers are far from this goal – even today's energy-efficient data centers consume almost half of their peak power when nearly idle [2].

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

Technologies that implement dynamic right-sizing are still far from standard in data centers due to a number of challenges. First, servers must be able to seamlessly transition into and out of power saving modes while not losing their state. There has been a growing amount of research into enabling this in recent years, dealing with virtual machine state [3], network state [4] and storage state [5], [6]. Second, such techniques must prove to be reliable, since administrators we talk to worry about wear-and-tear consequences of such technologies. Third, and the challenge that this paper addresses, it is unclear how to determine how many servers to toggle into power-saving mode and how to control servers and requests.

A goal of this paper is to provide a new algorithm to address this third challenge. To this end, 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).

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

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

Third, we validate our algorithm using two load traces (from Hotmail and a Microsoft Research data center) to evaluate the cost savings achieved via dynamic right-sizing in practice (Section V). We show that significant savings are possible under a wide range of settings and that savings become dramatic when the workload is predictable over an interval proportional to the toggling cost. The magnitude of the potential savings depend primarily on the peak-to-mean ratio (PMR) of the workload, with a PMR of 5 being enough to give 50% energy saving and 40% total cost saving even for quite bursty workloads. In the context of these real traces, we also discuss when it does and when it does not make sense to use dynamic right-sizing versus the alternative of 'valley-filling', i.e., using periods of low load to run background/maintenance tasks. We find that dynamic right-sizing provides more than 15% cost savings even when the background work makes up $> 40\%$ of the workload when the PMR is larger than 3.

II. Model formulation

We now describe the model we use to explore the cost savings possible via dynamic right-sizing. The assumptions used in the model are minimal and capture many properties of current data centers and traces we have obtained. Although we focus on single application environment, the model may be used to allocate resources to applications in a data center which hosts multiple applications using virtual machines.

A. 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 III and IV 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 V where λ_t is defined using real-world traces.

B. 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 [7]. 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 [8]. The combination of these models gives

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

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 the 99th percentile is $\log(100)/(1 - \lambda)$, and so the form of (1) does not change [8]. Similarly, when servers use dynamic speed scaling, if the energy cost is modeled as polynomial in speed as in [9], then the aggregate cost $f(\cdot)$ remains convex [10], [11]. Note that, in practice, $f(\cdot)$ can be empirically measured by observing the system over time.

[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 [2].

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 [4] or storage state [5], to several minutes (to migrate a large VM [3]). However, if (iii) is included, then β becomes on the order of the cost to run a server for an hour [12]. 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 paper, however previous work shows that solutions are possible [5].

C. 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:

$$\begin{aligned} \text{minimize} \quad & \sum_{t=1}^{T}\sum_{i=1}^{x_t} f(\lambda_{i,t}) + \beta\sum_{t=1}^{T}(x_t - x_{t-1})^+ \quad (2) \\ \text{subject to} \quad & 0 \le \lambda_{i,t} \le 1 \text{ and } \sum_{i=1}^{x_t}\lambda_{i,t} = \lambda_t, \end{aligned}$$

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 (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 (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 (2) into purely a capacity planning optimization:

$$\begin{aligned} \text{minimize} \quad & \sum_{t=1}^{T} x_t f(\lambda_t/x_t) + \beta\sum_{t=1}^{T}(x_t - x_{t-1})^+ \quad (3) \\ \text{subject to} \quad & x_t \ge \lambda_t \end{aligned}$$

It is this formulation of the data center optimization that we focus on for the remainder of the paper. 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) is a convex optimization problem for x_t. Throughout, denote the operating cost of a vector $X = (x_1, \dots, x_T)$ 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) 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, a bound of $N(t)$ servers can easily be imposed by defining the operating cost to be ∞ instead of $x_t f(\lambda_t / x_t)$ for $x_t > N(t)$. The results in this paper also apply to that model.

III. The optimal offline solution

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 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_τ^U and x_τ^L, respectively. Let $(x_{\tau,1}^L, \dots, x_{\tau,\tau}^L)$ be the solution vector to the following optimization problem

$$\begin{aligned} \text{minimize} \quad & \sum_{t=1}^{\tau} x_t f(\lambda_t / x_t) + \beta \sum_{t=1}^{\tau} (x_t - x_{t-1})^+ \\ \text{subject to} \quad & x_t \geq \lambda_t \end{aligned} \tag{4}$$

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

$$\begin{aligned} \text{minimize} \quad & \sum_{t=1}^{\tau} x_t f(\lambda_t / x_t) + \beta \sum_{t=1}^{\tau} (x_{t-1} - x_t)^+ \\ \text{subject to} \quad & x_t \geq \lambda_t \end{aligned} \tag{5}$$

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

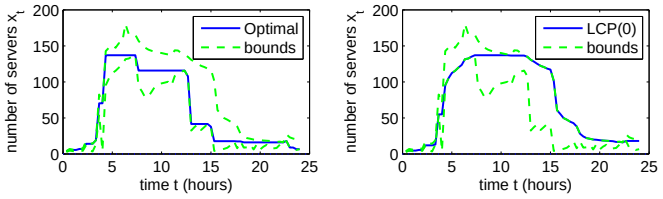
Notice that in each case, the optimization problem includes only times $1 \leq t \leq \tau$, and so ignores the arrival information for $t > \tau$. In the case of the lower bound, β cost is incurred for each server toggled on, while in the upper bound, β cost is incurred for each server toggled into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

Lemma 1. *For all τ, $x_\tau^L \leq x_\tau^* \leq x_\tau^U$.*

Given Lemma 1, we now characterize the optimal solution x_τ^*. Define $(x)_a^b = \max(\min(x, b), a)$ as the projection of x into $[a, b]$. Then, we have:

Theorem 1. *The optimal solution $X^* = (x_0^*, \dots, x_T^*)$ of the data center optimization problem (3) satisfies the following backward recurrence relation*

$$x_\tau^* = \begin{cases} 0, & \tau \geq T; \\ (x_{\tau+1}^*)_{x_\tau^L}^{x_\tau^U}, & \tau \leq T-1. \end{cases} \tag{6}$$



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

Fig. 1. Illustrations of (a) the offline optimal solution and (b) LCP(0) for the first day of the MSR workload described in Section V with a sampling period of 20 minutes. The operating cost is defined by (1) with $d_0 = 1.5$, $d_1 = 1$, $\mu = 1$, $e_0 = 1$ and $e_1 = 0$ and the switching cost has $\beta = 8$.

Theorem 1 and Lemma 1 are proven in Appendix A.

An example of the optimal x_t^* can be seen in Figure 1(a). Many more numeric examples of the performance of the optimal offline algorithm are provided in Section V.

Theorem 1 and Figure 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.

IV. Lazy capacity provisioning

A major contribution of this paper 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 V 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 III. Like the optimal solution, it "lazily" stays within upper and lower bounds. However, it does this moving forward in time instead of backwards in time.

Before defining LCP(w) formally, recall that the bounds x_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 (5) and $x_\tau^{L,w} = x_{\tau+w,\tau}^L$ in that of (4). Note that $x_\tau^{U,0} = x_\tau^U$ and $x_\tau^{L,0} = x_\tau^L$. The following generalization of Lemma 1 is proven in Appendix B.

Lemma 2. *$x_\tau^L \leq x_\tau^{L,w} \leq x_\tau^* \leq x_\tau^{U,w} \leq x_\tau^U$ for all $w \geq 0$.*

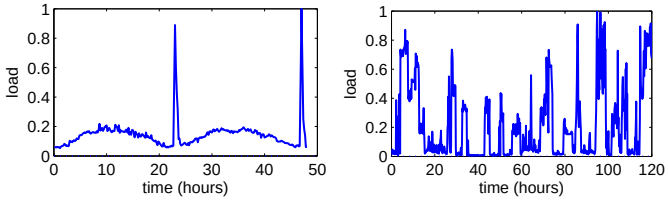
Now, we are ready to define LCP(w) using $x_\tau^{U,w}$ and $x_\tau^{L,w}$.

Algorithm 1. ***Lazy Capacity Provisioning, LCP(w).*** *Let $X^{LCP(w)} = (x_0^{LCP(w)}, \dots, x_T^{LCP(w)})$ denote the vector of active servers under LCP(w). This vector can be calculated using the following forward recurrence relation*

$$x_\tau^{LCP(w)} = \begin{cases} 0, & \tau \leq 0; \\ (x_{\tau-1}^{LCP(w)})_{x_\tau^L}^{x_\tau^{U,w}}, & \tau \geq 1. \end{cases} \tag{7}$$

Figure 1(b) illustrates the behavior of LCP(0). Note its similarity with Figure 1(a), but with the laziness in forward time instead of reverse time.

The computational demands of LCP(w) may initially seem prohibitive as τ grows, since calculating $x_\tau^{U,w}$ and $x_\tau^{L,w}$



(a) Hotmail (b) MSR

Fig. 2. Illustration of the traces used for numerical experiments.

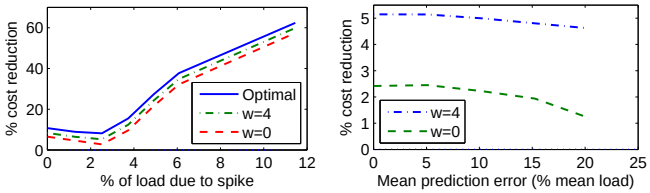


Fig. 3. Impact of overnight peak in the Hotmail workload.

Fig. 4. Impact of prediction error on LCP(w) under Hotmail workload.

requires solving convex optimizations of size $\tau + w$. However, it is possible to calculate $x_\tau^{U,w}$ and $x_\tau^{L,w}$ without using the full history. Lemma 9 in Appendix B implies that it is enough to use only the history since the most recent point when the solutions of (4) and (5) are either both increasing or both decreasing, if such a point exists. In practice, this period is typically less than a day due to diurnal traffic patterns, and so the convex optimization, and hence LCP(w), remains tractable even as τ grows.

Next, consider the cost incurred by LCP(w). Section V discusses the cost in realistic settings, while in this section we focus on worst-case bounds. In particular, we derive a competitive ratio. We say that an algorithm is *C-competitive* if for all problem instances (all convex cost functions and finite arrival rate vectors), the cost of the algorithm is less than C times the cost of the optimal offline solution. The following theorem is proven in Appendix B.

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

V. Case studies

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 and energy savings that come from dynamic right-sizing in data centers. To accomplish these goals, we experiment using two real-world traces.

A. Experimental setup

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

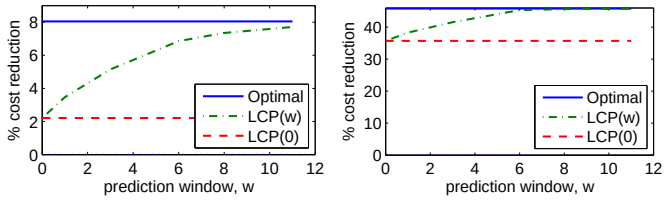
Cost benchmark: Current data centers typically do not use dynamic right-sizing and so to provide a benchmark against which LCP(w) is judged, we consider the cost incurred by a 'static' right-sizing scheme for capacity provisioning. This chooses a constant number of servers that minimizes the costs incurred based on full knowledge of the entire workload. This policy is clearly not possible in practice, but it provides a 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.

Cost function parameters: The cost is characterized by the four parameters of (1), d_0, d_1, e_0 and e_1, and the switching cost β. We choose units such that the fixed energy cost is $e_0 = 1$. The load-dependent energy consumption is set to $e_1 = 0$, because the energy consumption of current servers is dominated by the fixed costs [2].

The delay cost d_1 reflects revenue lost due to customers being deterred by delay, or to violation of SLAs. We set $d_1/e_0 = 1$ for most experiments but consider a wide range of settings in Figure 7. The minimum perceptible delay is set to $d_0 = 1.5$ times the time to serve a single job. The value 1.5 is realistic or even conservative, since "valley filling" experiments similar to those of Section V-B show that a smaller value would result in a significant added cost when valley filling, which operators now do with minimal incremental cost.

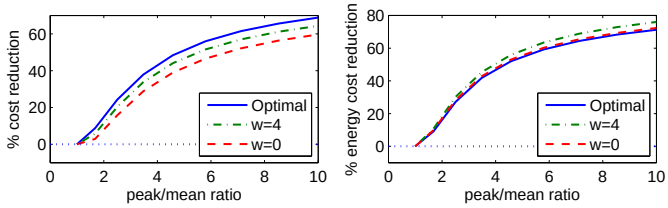
The normalized switching cost β/e_0 measures the duration a server must be powered down to outweigh the switching cost. We use $\beta = 6$, which corresponds to the energy consumption for one hour (six samples). This was chosen as an estimate of the time a server should sleep so that the wear-and-tear of power cycling matches that of operating [12].

Workload information: The workloads for these experiments are drawn from two real-world data center traces. The first set of traces is from Hotmail, a large email service running on tens of thousands of servers. We used I/O traces from 8 such servers over a 48-hour period, starting at midnight (PDT) on Monday August 4 2008 [5]. The second set of I/O traces is taken from 6 RAID volumes at MSR Cambridge. The traced period was 1 week starting from 5PM GMT on the 22nd February 2007 [5]. Thus, these activity traces represent a service used by millions of users and a small service used by hundreds of users. The traces are normalized to the peak load, which are shown in Figure 2. Both sets of traces show strong diurnal properties and have peak-to-mean ratios (PMRs) of 1.64 and



(a) Hotmail (b) MSR

Fig. 5. Impact of prediction window size on cost incurred by LCP(w).



(a) Total cost (b) Energy cost

Fig. 6. Impact of the peak-to-mean ratio of the workload on the total cost and energy cost incurred by LCP(w) in the Hotmail workload.

4.64 for Hotmail and MSR respectively. Loads were averaged over disjoint 10 minute intervals.

The Hotmail trace contains significant nightly activity due to maintenance processes (backup, index creation etc). The data center, however, is provisioned for the peak foreground activity. This creates a dilemma: should our experiments include the maintenance activity or to remove it? Figure 3 illustrates the impact of this decision. If the spike is retained, it makes up nearly 12% of the total load and forces the static provisioning to use a much larger number of servers than if it were removed, making savings from dynamic right-sizing much more dramatic. To provide conservative estimates of the savings from right-sizing, we chose to trim the size of the spike to minimize the savings from right-sizing.

Prediction error: The LCP(w) algorithm depends on having estimates for the arrival rate during the current timeslot as well as for w timeslots into the future. Our analysis in Section IV assumes that these estimates are perfect, but of course in practice there are prediction errors. However, Figure 4 shows that LCP(w) is fairly robust to prediction errors, and so this assumption is not problematic. In particular, the cost reduction on the Hotmail trace, relative to a static scheme with full, perfect, knowledge of the workload is still significant when additive white Gaussian noise of increasing variance is added to predictions used by LCP(w). The plot for the MSR trace is qualitatively the same, however the cost savings is actually significantly larger. Given that prediction errors for real data sets tend to be small [13], [14], based on these plots, to simplify our experiments we allow LCP(w) perfect predictions.

B. When is right-sizing beneficial?

Our experiments are organized in order to illustrate the impact of a wide variety of parameters on the cost-savings provided by dynamic right-sizing via LCP(w). The goal is to better understand when dynamic right-sizing can provide large enough cost-savings to warrant the extra implementation complexity. Remember that throughout, we have attempted to choose experimental settings so that the benefit of dynamic right-sizing is conservatively estimated.

Impact of prediction window size: The first parameter we study is the impact of the predictability of the workload. In particular, depending on the workload, the prediction window w for which accurate estimates can be made could be on the order of tens of minutes or on the order of hours. Figure 5 illustrates the impact this has on the cost savings of LCP(w), where the unit of w is one timeslot which is 10 minutes.

The first observation from Figure 5 is that the savings possible in the MSR trace are larger than in the Hotmail trace. However, in both cases, 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.

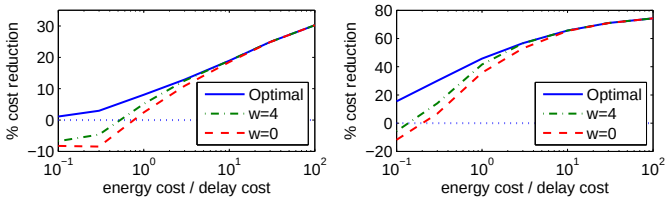
Impact of peak-to-mean ratio (PMR): Dynamic right-sizing inherently exploits the gap between the peaks and valleys of the workload, and intuitively provides larger savings as that gap grows. Figure 6 illustrates that this intuition holds for both cost savings and energy savings. The gain grows quickly from zero at PMR=1, to 5–10% at PMR$\approx$ 2 which is common in large data centers, to very large values for the higher PMRs common in small to medium sized data centers. This shows that, even for small data centers where the overhead of implementing right-sizing is amortized over fewer servers, there is a significant benefit in doing so. To provide some context for the monetary value of these savings, consider that a typical 50,000 server data center has an electricity bill of around $1 million/month [1].

The workload for the figure is generated from the Hotmail 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 6 includes only the results for Hotmail, the resulting plot for the MSR trace is nearly identical. This highlights that the difference in cost savings observed between the two traces is primarily due to the fact that the PMR of the MSR trace is so much larger than that of the Hotmail trace.

Impact of energy costs: Clearly the benefit of dynamic right-sizing is highly dependent on the cost of energy. As the economy is forced to move towards more expensive renewable energy sources, this cost will inevitably increase and Figure 7 shows how this increasing cost will affect the cost savings possible from dynamic right-sizing. Note that the cost savings from dynamic right-sizing grow quickly as energy costs rise. However, even when energy costs are quite small relative to delay costs, we see improvement in the case of the MSR workload due to its large PMR.

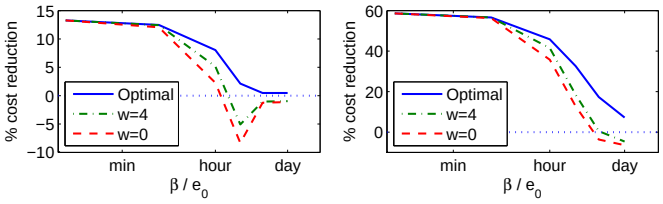
Impact of switching costs: One of the main worries when considering right-sizing is the switching cost of toggling servers, β, which includes the delay costs, energy costs, costs of wear-and-tear, and other risks involved. Thus, an important question to address is: "How large must switching costs be before the cost savings from right-sizing disappears?"

Figure 8 shows that significant gains are possible provided β is smaller than the duration of the valleys. Given that the energy costs, delay costs, and wear-and-tear costs are likely to be on the order of an hour, this implies that unless the risks associated with toggling a server are perceived to be extreme, the benefits from dynamic right-sizing are large in the MSR trace (high PMR case). Though the gains are smaller in the Hotmail case for large β, this is because the spike



(a) Hotmail (b) MSR

Fig. 7. Impact of increasing energy costs.



(a) Hotmail (b) MSR

Fig. 8. Impact of switching cost, against time on a logarithmic scale.

of background work splits an 8 hour valley into two short 4 hour valleys. If these tasks were shifted or balanced across the valley, the Hotmail trace would show significant cost reduction for much larger β, similarly to the MSR trace.

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

Figure 9 shows that, in fact, dynamic right-sizing provides cost savings even when background work makes up a significant fraction of the total load. For the Hotmail trace, significant savings are still possible when background load makes upwards of 10% of the total load, while for the MSR trace this threshold becomes nearly 60%. Note that Figure 9 results from considering 'ideal' valley filling, which results in a perfectly flat load during the valleys, but does not give background processes lower queueing priority.

VI. Related work

This paper is not alone in approaching the task of developing algorithms for dynamic right-sizing. Interest in right-sizing has been growing since [15] and [16] appeared at the start of the decade. Approaches range from very "analytic" work focusing on developing algorithms with provable guarantees to "systems" work focusing purely on implementation. Early systems work such as [16] achieved substantial savings despite ignored switching costs in their design. Other designs have focused on decentralized frameworks, e.g., [17] and [18], as opposed to the centralized framework considered here. A recent survey is [19].

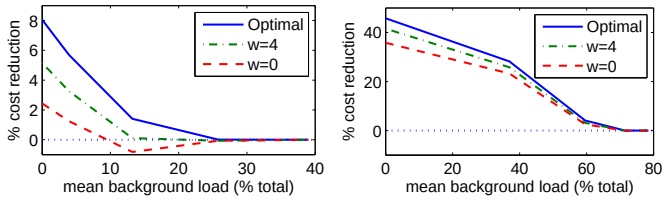
Related analytic work focusing on dynamic right-sizing includes [20], which reallocates resources between tasks within a data center, and [21], which considers sleep of individual components, among others. Typically, approaches have applied optimization using queueing theoretic models, e.g., [22], [23], or control theoretic approaches, e.g., [24]–[26]. A recent survey of analytic work focusing on energy efficiency in general is [27]. Our work is differentiated from this literature by the generality of the model considered, which subsumes most common energy and delay cost models used by analytic researchers, and the fact that we provide worst-case guarantees for the cost of the algorithm, which is typically not possible for queueing or control theoretic based algorithms. For example, using 'model predictive control' [28] has often been suggested for dynamic right-sizing, but has an unbounded competitive ratio for the model considered here.

The model and algorithm introduced in this paper most closely ties to the online algorithms literature, specifically the classic rent-or-buy (or "ski rental") problem [29]. The optimal deterministic strategy for deciding when to turn off a single idle server (i.e., to stop 'renting' and 'buy') is the 2-competitive [30]. Additionally, there is a randomized algorithm which is asymptotically $e/(e-1)$-competitive [31]. Both of these competitive ratios extend to the more general setting of putting a server in one of several sleep modes, with different power consumption and transition costs [21]. An important difference between these simple models and the current paper is that the cost of the 'buy' and 'rent' actions may change arbitrarily over time in the data center optimization problem. Problems with this sort of dynamics typically have competitive ratios greater than 2. For example, when rental prices vary in time, the competitive ratio is unbounded in general [32]. Further, for "metrical task systems" [33]–[35], which generalize rent-or-buy problems and the data center optimization problem, there are many algorithms available, but they typically have competitive ratios that are poly-logarithmic in the number of servers. Perhaps the most closely related prior work from this area is the page-placement problem (deciding on which server to store a file), which has competitive ratio 3 [36]. The page replacement-problem is nearly a discrete version of the data center optimization problem where the cost function is restricted to $f(x) = |x - 1|$. Finally, it is important to note that LCP(w) is quite different than the classical algorithms applied for rent-or-buy problems.

VII. Summary and concluding remarks

This paper has provided a new online algorithm, LCP(w), for dynamic right-sizing in data centers. The algorithm is motivated by the structure of the optimal offline solution and guarantees cost no larger than 3 times the optimal cost, under very general settings – arbitrary workloads, general delay cost models, and general energy cost models. Further, in realistic settings the cost of LCP(w) is nearly optimal. Additionally, LCP(w) is simple to implement in practice and does not require significant computational overhead.

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



(a) Hotmail (b) MSR

Fig. 9. Impact of background processes. The improvement of LCP(w) over static provisioning as a function of the percentage of the workload that is background tasks.

REFERENCES

[1] J. Hamilton, "Cost of power in large-scale data centers," http://perspectives.mvdirona.com/, Nov. 2009.

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

[3] 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*, 2005, pp. 273–286.

[4] 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.

[5] E. Thereska, A. Donnelly, and D. Narayanan, "Sierra: a power-proportional, distributed storage system," Microsoft Research, Tech. Rep. MSR-TR-2009-153, 2009.

[6] 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.

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

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

[9] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. IEEE FOCS*, 1995, pp. 374–382.

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

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

[12] 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," University of California at Berkeley, Tech. Rep. UCB/EECS-2008-127, 2008.

[13] 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, Sep. 2007.

[14] 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*, vol. 12, no. 1, pp. 1–15, Mar. 2009.

[15] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *Proc. ACM SOSP*, 2001, pp. 103–116.

[16] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath, "Load balancing and unbalancing for power and performacne in cluster-based systems," in *proc. Compilers and Operating Systems for Low Power*, 2001.

[17] B. Khargharia, S. Hariri, and M. Yousif, "Autonomic power and performance management for computing systems," *Cluster computing*, vol. 11, no. 2, pp. 167–181, Dec. 2007.

[18] A. Kansal, J. Liu, A. Singh, , R. Nathuji, and T. Abdelzaher, "Semantic-less coordination of power management and application performance," in *ACM SIGOPS*, 2010, pp. 66–70.

[19] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," Univ. of Melbourne, Tech. Rep. CLOUDS-TR-2010-3, 2010.

[20] C. G. Plaxton, Y. Sun, M. Tiwari, , and H. Vin, "Reconfigurable resource scheduling," in *ACM SPAA*, 2006.

[21] S. Irani, R. Gupta, and S. Shukla, "Competitive analysis of dynamic power management strategies for systems with multiple power savings states," in *Proc. Design, Automation, and Test in Europe*, 2002, p. 117.

[22] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," in *Proc. of ACM Sigmetrics*, 2009.

[23] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Performance Evaluation*, vol. 67, no. 11, pp. 1155 – 1171, 2010.

[24] T. Horvath and K. Skadron, "Multi-mode energy management for multi-tier server clusters," in *Proc. PACT*, 2008.

[25] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *Proc. Sigmetrics*, 2005.

[26] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Proc. IEEE/IFIP NOMS*, Apr. 2010.

[27] S. Albers, "Energy-efficient algorithms," *Comm. of the ACM*, vol. 53, no. 5, pp. 86–96, 2010.

[28] E. F. Camacho and C. Bordons, *Model predictive control*. Springer, 2004.

[29] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press, 1998.

[30] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, "Competitive snoopy caching," *Algorithmica*, vol. 3, no. 1, pp. 77–119, 1988.

[31] A. R. Karlin, C. Kenyon, and D. Randall, "Dynamic tcp acknowledgement and other stories about $e/(e-1)$," in *Proc. ACM STOC*, 2001.

[32] M. Bienkowski, "Price fluctuations: To buy or to rent," in *Approximation and Online Algorithms*, 2008, pp. 25–36.

[33] A. Borodin, N. Linial, and M. E. Saks, "An optimal on-line algorithm for metrical task system," *J. ACM*, vol. 39, no. 4, pp. 745–763, 1992.

[34] A. Blum and C. Burch, "On-line learning and the metrical task system problem," *Machine Learning*, vol. 39, no. 1, Apr. 2000.

[35] A. Fiat and M. Mendel, "Better algorithms for unfair metrical task systems and applications," in *Proc. STOC*, 2000, pp. 725–734.

[36] D. L. Black and D. D. Sleator, "Competitive algorithms for replication and migration problems," Carnegie Mellon University, Tech. Rep. CMU-CS-89-201, 1989.

APPENDIX A
ANALYSIS OF THE OFFLINE OPTIMAL SOLUTION

In this section we will prove Lemma 1 and Theorem 1. Before beginning the proofs, let us first rephrase the data center optimization (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) that $x_t \geq \lambda_t$ and the rephrasing as:

$$\begin{aligned} \text{minimize} \quad & \sum_{t=1}^{T} x_t f(\lambda_t/x_t) + \beta \sum_{t=1}^{T} y_t \\ \text{subject to} \quad & y_t \geq x_t - x_{t-1}, \text{ and } y_t \geq 0. \end{aligned} \tag{8}$$

Next, we want to work with the dual of optimization (8). Let $F^*_{\lambda_t}(z)$ be the conjugate of the function $F_{\lambda_t}(z) = zf(\lambda_t/z)$. Then the corresponding dual problem of (8)

$$\begin{aligned} \text{max.} & -\sum_{t=1}^{T-1} F^*_{\lambda_t}(\mu_{t+1} - \mu_t) - F^*_{\lambda_T}(-\mu_T) - \mu_1 x_0 \\ \text{subject to } & 0 \leq \mu_t \leq \beta, \end{aligned} \tag{9}$$

where the complementary slackness conditions are

$$\mu_t(x_t - x_{t-1} - y_t) = 0 \tag{10}$$

$$(\beta - \mu_t) y_t = 0, \tag{11}$$

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 (8) and the upper and lower bounds, i.e., optimizations (5) and (4). Specifically, if $\mu_{\tau+1} = 0$ in the solution of optimization (9), then $\mu_1, \ldots, \mu_\tau$ is the solution to the following problem:

$$\begin{aligned} \text{max.} & -\sum_{t=1}^{\tau-1} F^*_{\lambda_t}(\mu_{t+1} - \mu_t) - F^*_{\lambda_\tau}(-\mu_\tau) - \mu_1 x_0 \\ \text{subject to } & 0 \leq \mu_t \leq \beta. \end{aligned} \tag{12}$$

Thus, the corresponding $x_1, \ldots, x_\tau$ is the solution to optimization (4). On the other hand, if $\mu_{\tau+1} = \beta$ in the solution

of optimization (9), then $\mu_1, \dots, \mu_\tau$ is the solution to the following problem:

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

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

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

It is now easy to check that the corresponding $x_1, \dots, x_\tau$ is the solution to optimization (5).

We require two technical lemmas before moving to the proofs of Lemma 1 and Theorem 1. We omit the proofs.

Lemma 3. *Given $\Lambda = (\lambda_i, \dots, \lambda_j)$, $x_{i-1} = \hat{x}_{i-1}$ and $x_j \le \hat{x}_j$, Let $X = (x_i, \dots, x_{j-1})$ be the solution to optimization (4) given Λ with constraints fixing the initial and final values as x_{i-1} and x_j respectively. Let $\hat{X} = (\hat{x}_i, \dots, \hat{x}_{j-1})$ be the solution to optimization (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 $\hat{X} \ge X$.*

Lemma 4. *Given $\lambda_1, \dots, \lambda_\tau$ and additional constraints $x_0 = x_S$ and $x_\tau = x_E$, optimization (4) and optimization (5) yield the same solution vector.*

We now complete the proofs of Lemma 1 and Theorem 1.

Proof of Lemma 1: We begin with some definitions. Let $X^L_\tau = (x^L_{\tau,1}, x^L_{\tau,2}, \dots, x^L_{\tau,\tau})$ be the solution of optimization (4) at time τ and define X^U_τ symmetrically. Additionally, let X^*_τ be the solution to optimization (3) with arrival rates $(\lambda_1, \dots, \lambda_\tau, 0)$ and constraint $x_{\tau+1} = x^*_{\tau+1} \ge 0$.

Now, consider the optimization problem (3) with arrival rate $(\lambda_1, \dots, \lambda_\tau, 0)$ and constraint $x_{\tau+1} = 0$ and denote its solution by $\{x'_t\}$. Since $x'_\tau \ge \lambda_\tau > 0$, the complementary slackness condition gives that $\mu_{\tau+1} = 0$ in its dual problem. Thus, $\mu'_1, \dots, \mu'_\tau$ is the solution to optimization (12), and the corresponding $x'_1, \dots, x'_\tau$ is the solution to optimization (4), i.e., $x'_t = X^L_{\tau,t}$. Next, we apply Lemma 3, which gives that $x^L_\tau \le x^*_\tau$, as desired.

Symmetrically to the argument for the lower bound, together with Lemma 4, we can see that X^U_τ is the solution to optimization (3) with arrival rates $(\lambda_1, \dots, \lambda_\tau, 0)$ and constraint $x_{\tau+1} = \infty$. Further, X^*_τ is the solution of optimization (3) with arrival rates $(\lambda_1, \dots, \lambda_\tau, 0)$ and constraint $x_{\tau+1} = x^*_{\tau+1} < \infty$. Again, applying Lemma 3 gives that for all t, $x^*_{\tau,t} \le x^U_{\tau,t}$, and thus, in particular, we have that $x^*_\tau \le x^U_\tau$, as desired. ∎

Proof of Theorem 1: As a result of Lemma 1, we know that $x^*_\tau \in [x^L_\tau, x^U_\tau]$ for all τ. Further, if $x^*_\tau > x^*_{\tau+1}$, by the complementary slackness condition (10), we have that $\mu_{\tau+1} = 0$. Thus, in this case, x^*_τ solves optimization (4) for the lower bound, i.e., $x^*_\tau = x^L_\tau$. Symmetrically, if $x^*_\tau < x^*_{\tau+1}$, we have that complementary slackness condition (11) gives $\mu_{\tau+1} = \beta$ and so x^*_τ solves optimization (5) for the upper bound, i.e., $x^*_\tau = x^U_\tau$. Thus, whenever x^*_τ is increasing/decreasing it must be matching the upper/lower bound, respectively. ∎

APPENDIX B
ANALYSIS OF LAZY CAPACITY PROVISIONING, LCP(w)

In this section we will prove Lemma 2 and Theorem 2. We begin by proving Lemma 2.

Proof of Lemma 2: First, we prove that $x^{L,w}_\tau \le x^*_\tau$. By definition, $x^{L,w}_\tau = x^L_{\tau+w,\tau}$, and so is the solution to optimization (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 (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 3, to conclude that $x^{L,w}_\tau \le x^*_\tau$.

Next, we prove that $x^L_\tau \le x^{L,w}_\tau$. To see this we notice that $x^{L,w}_\tau$ is also the solution to optimization (4) given $\lambda_1, \dots, \lambda_\tau, 0, \dots$ and the added (redundant) constraint that $x_{\tau+1} = x^L_{\tau+w,\tau+1} \ge 0$. Then, x^L_τ is the solution to optimization (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 3, to conclude that $x^L_\tau \le x^{L,w}_\tau$.

Finally, the proof that $x^*_\tau \le x^{U,w}_\tau \le x^U_\tau$ 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 1. *The optimal solution of the data center optimization (3) satisfies the following backwards recurrence relation*

$$x^*_\tau = \begin{cases} 0, & \tau \ge T; \\ (x^*_{\tau+1})^{x^{U,w}_\tau}_{x^{L,w}_\tau}, & \tau \le T-1. \end{cases} \tag{13}$$

Moving to the proof of Theorem 2, the first step is to use the above lemmas to characterize the relationship between $x^{LCP(w)}_\tau$ and x^*_τ. Note that $x^{LCP(w)}_0 = x^*_0 = x^{LCP(w)}_T = x^*_T = 0$.

Lemma 5. *Consider timeslots $0 = t_0 < t_1 < \ldots < t_m = T$ such that $x^{LCP(w)}_{t_i} = x^*_{t_i}$. Then, during each segment (t_{i-1}, t_i), either*

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

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

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

Now we must consider two cases. First, consider the case that $x^{LCP(w)}_{t_i+1} > x^*_{t_i+1}$. It is easy to see that $x^{LCP(w)}_{t_i+1}$ doesn't match the lower bound since $x^*_{t_i+1}$ is not less than the lower bound. Note, we have that $x^{LCP(w)}_{t_i} \ge x^{LCP(w)}_{t_i+1}$ since, by definition, LCP(w) will never choose to increase the number of servers it uses unless it matches the lower bound. Consequently, it must be that $x^*_{t_i} = x^{U,w}_{t_i} \ge x^{LCP(w)}_{t_i+1} > x^*_{t_i+1}$. Both $x^{LCP(w)}_{t_i}$ 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 $x^{L,w}_t$ 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 5, we bound the switching cost of LCP(w).

Lemma 6. $cost_s(X^{LCP(w)}) = cost_s(X^*)$.

Proof: Consider the sequence of times $0 = t_0 < t_1 < \ldots < t_m = T$ such that $x_{t_i}^{LCP(w)} = x_{t_i}^*$ identified in Lemma 5. Then, each segment (t_{i-1}, t_i) starts and ends with the same number of servers being used under both LCP(w) and the optimal solution. Additionally, the number of servers is monotone for both LCP(w) and the optimal solution, thus the switching cost incurred by LCP(w) and the optimal solution during each segment is the same. ■

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

Lemma 7. $cost_o(X^{LCP(w)}) \leq cost_o(X^*) + \beta \sum_{t=1}^{T} |x_t^* - x_{t-1}^*|$.

Proof: Consider the sequence of times $0 = t_0 < t_1 < \ldots < t_m = T$ such that $x_{t_i}^{LCP(w)} = x_{t_i}^*$ identified in Lemma 5, and consider specifically one of these intervals (t_{i-1}, t_i) such that $x_{t_{i-1}}^{LCP(w)} = x_{t_{i-1}}^*$, $x_{t_i}^{LCP(w)} = x_{t_i}^*$.

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

We handle *case (i)* first. Define $X_\tau = (x_{\tau,1}, \ldots, x_{\tau,\tau})$ as the solution vector of optimization (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 (5) on $\lambda_1, \ldots, \lambda_\tau, 0$ with the additional constraint that $x_{\tau+1} = x_\tau^{LCP(w)}$. Note that Lemma 4 gives that $X'_{\tau+1}$ is also the solution vector to optimization (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

$$\sum_{t=t_{i-1}+1}^{t_i} x_t^{LCP(w)} f(\lambda_t / x_t^{LCP(w)}) \leq \sum_{t=t_{i-1}+1}^{t_i} x_t^* f(\lambda_t / x_t^*) + \beta |x_{t_{i-1}} - x_{t_i}|. \quad (14)$$

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 (10) would imply that $\mu_{\tau+1} = 0$ for the optimization (4). Thus $x'_{\tau+1,\tau}$ would belong to the optimal solution of (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

$$c(X'_{\tau+1}) = c(X_\tau) + x_\tau^{LCP(w)} f(\lambda_{\tau+1} / x_\tau^{LCP(w)}). \quad (15)$$

Next, recalling $x_\tau^{LCP(w)}$ is non-increasing in case (i) by Lemma 5, 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 3 that $X_{\tau+1} \leq X'_{\tau+1}$ and thus $x_{\tau+1,\tau} \leq x_\tau^{LCP(w)}$. Therefore, we have:

$$\begin{aligned} c(X'_{\tau+1}) &\leq c((x_{\tau+1,1}, \ldots, x_{\tau+1,\tau}, x_\tau^{LCP(w)})) \\ &= c((x_{\tau+1,1}, \ldots, x_{\tau+1,\tau})) \\ &\quad + x_\tau^{LCP(w)} f(\lambda_{\tau+1} / x_\tau^{LCP(w)}). \end{aligned} \quad (16)$$

Combining equations (15) and (16), 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 (14), 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 (14) holds in this case as well.

To complete the proof we combine the results from case (i) and case (ii), summing equation (14) over all segments (and the additional times when $x_t^{LCP(w)} = x_t^*$). ■

We can now prove the competitive ratio in Theorem 2.

Lemma 8. $cost(X^{LCP(w)}) \leq cost(X^*) + 2cost_s(X^*)$. *Thus, LCP(w) is 3-competitive for the data center optimization (3).*

Proof: Combining Lemma 7 and Lemma 6 gives that $cost(X^{LPC(w)}) \leq cost(X^*) + \beta |x_t^* - x_{t-1}^*|$. Note that, because both LCP(w) and the optimal solution start and end with zero servers on, we have $\sum_{t=1}^{T} |x_t^* - x_{t-1}^*| = 2 \sum_{t=1}^{T} (x_t^* - x_{t-1}^*)^+$, which completes the proof. ■

All that remains for the proof of Theorem 2 is to prove that the competitive ratio of 3 is tight. Space constraint prevent including the full proof, however the instance which provides the tight lower bound is defined as follows. The cost function is defined as $f(z) = z^m + f_0$ for $(0 \leq z \leq 1, m \geq 2)$ and $\beta = 0.5$. The arrival rate at time t is $\lambda_t = \delta^{t-1}$ with $\delta > 1$ for $1 \leq t \leq n$, and $\lambda_t = 0$ for $n+1 \leq t \leq T$, where $n = \lceil \log_\delta \frac{1}{\delta - 1} \rceil$. Further, $f_0 = \frac{\beta(\delta^m - 1)}{n(\delta^{mn} - 1)}$ and $T > \beta / f_0 + n$. The global optimal solution and the solution given by $LPC(w)$ can be calculated analytically, and yield that $cost(X^{LPC(w)}) / cost(X^*) \to 3 - 2/m$ as $\delta \to 1$. For large m, the competitive ratio is arbitrarily close to 3.

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

Lemma 9. *If there exists an index $t \in [1, \tau - 1]$ such that $x_{\tau,t+1}^U < x_{\tau,t}^U$ or $x_{\tau,t+1}^L > x_{\tau,t}^L$, then $(x_{\tau,1}^U, \ldots, x_{\tau,t}^U) = (x_{\tau,1}^L, \ldots, x_{\tau,t}^L)$, 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 $x_{\tau,t}^U$ and the second over $[t+1, \tau']$ with initial condition $x_{\tau,t}^U$.*