Online Dynamic Capacity Provisioning in 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 implementing cloud services, yet much of that power is used to maintain excess service capacity during periods of low load. In this work, we study how to avoid such waste via an online dynamic capacity provisioning. We overview recent results showing that the optimal offline algorithm for dynamic capacity provisioning has a simple structure when viewed in reverse time, and this structure can be exploited to develop a new 'lazy' online algorithm which is 3-competitive. Additionally, we analyze the performance of the more traditional approach of receding horizon control and introduce a new variant with a significantly improved worst-case performance guarantee.**

I. Introduction

Energy costs represent a significant fraction of a data center's budget [1] and this fraction is expected to grow. Hence, there is a growing push to improve the energy efficiency of the data centers. A promising approach for making data centers more energy efficient is using software to dynamically 'right-size' the data center, i.e., adapt the dispatching so that during periods of low load some servers are allowed to enter a power-saving mode (e.g., go to sleep or shut down).

However, entering and leaving sleep mode incurs a penalty ("switching cost"), in terms of latency, energy consumption, or wear-and-tear. This means that decisions to sleep cannot be made independently at different time instants. The problem is challenging due to the lack of knowledge about future workloads, which means that a server that is put to sleep may soon need to be woken again. There is a significant and growing literature on this topic [2]–[5].

This paper provides an overview of recent results providing online algorithms to decide the provisioning at each time instant without or with a little information of future workload. To this end, we discuss a simple but general model that captures the major issues of right-sizing. With this model, we first analytically characterize the optimal offline solution (Section III-A). We show that it exhibits a simple, 'lazy' structure when viewed in reverse time. Second, we discuss a novel, practical online algorithm motivated by this structure (Section III-B). The algorithm, named *Lazy Capacity Provisioning* (LCP), mimics the 'lazy' structure of the optimal algorithm, but proceeding forwards instead of backwards in time. Importantly, LCP is 3-competitive, i.e., its cost is at most 3 times that of the optimal offline solution. Third, we analyze the traditional approach of *Receding Horizon Control* (RHC). We show that RHC performs well when servers are homogeneous; specifically, it has performance that quickly tends toward optimality as the prediction window increases. However, we also show that RHC can perform badly when servers are heterogeneous, regardless of the length of the prediction window. To address this issue, we discuss a variant of RHC that is guaranteed to perform well in heterogeneous settings. Specifically, under both homogeneous and heterogeneous settings, their competitive ratio matches that of RHC in the homogeneous setting. Moreover, we validate our algorithm using two real traces (Section V). We show that significant savings are possible under a wide range of settings.

All the results described in this paper are proven and discussed in more detail in [3] and [6].

II. Model

Our focus is on understanding how to dynamically provision the (active) service capacity of a large, possibly heterogeneous pool of servers so as to minimize the "cost" of the system, which may include both energy and quality of service.

A. Workload model

We consider $S \geq 1$ types of servers, each of which has a different cost for serving different types of jobs, and $J \geq 1$ types of jobs. We take a discrete-time model where the timeslot length matches the timescale at which servers can enter or leave power saving states. There is a (possibly long) time-interval of interest $t \in \{1, \ldots, T\}$. The mean request rate for timeslot t is denoted by $\lambda_t = (\lambda_{t,j})_{j \in J}$, where $\lambda_{t,j}$ is the mean request rate (arrival rate) for type j jobs at time t. We set $\lambda_t = 0$ for $t < 1$ and $t > T$, and assume that jobs are short so that work does not carry over between slots and provisioning can be based on the average arrival rate during a timeslot. In the data center setting, T could be a year, a timeslot could be 10 minutes, and a job length could be on a second or less.

B. Cost model

Our goal is to provide insight into the important decision: determining the number of active servers, $x_t = (x_{t,s})_{s \in S}$, where $x_{t,s}$ is the number of active servers of type s during timeslot t.

We decompose the cost incurred by the system into two components: (i) the *operating cost* incurred by using active servers to serve requests in each timeslot. (ii) the *switching cost* incurred by changing provisioning between timeslots. Note that both components may include costs of energy, delay, and even wear-and-tear.

To model the operating costs, we use a (possibly time varying) function $f_t(x_t, \lambda_t)$ for each timeslot, which represents the cost of using $x_t = (x_{t,1}, \ldots, x_{t,S})$ servers to serve arriving requests $\lambda_t = (\lambda_{t,1}, \ldots, \lambda_{t,J})$ under the optimal dispatching of λ_t over x_t. Note that the optimal dispatching can be computed easily in many cases, such as when each f_t is convex in x_t. Thus, we do not explicitly consider the dispatching decision in the following and simply assume it is performed optimally.

The switching costs are modeled by a function $d(x_{t-1}, x_t)$, which represents the cost of changing the number of active servers of each type from vector x_{t-1} to vector x_t. Let β^+ and β^- be vectors such that β_s^+ is the cost of turning a server of type s on, and β_s^- is the cost of turning it off. Then, it is natural to use

$$d(x_{t-1}, x_t) = \beta^+ \cdot (x_t - x_{t-1})^+ + \beta^- \cdot (x_{t-1} - x_t)^+,$$

where $(x)^+ = \max(0, x)$ elementwise.

C. Cost optimization problem

Given the workload and cost models above, the system goal is to choose the active number of servers x_t so as to minimize the total cost during $[1, T]$. We assume $x_0 = 0$ and $x_{T+1} = 0$, then the number of times a server is turned on in $[1, T]$ is equal to the number of times it is turned off in $[2, T+1]$. Thus the optimization depends on β^+ and β^- only through their sum $\beta = \beta^+ + \beta^-$. Therefore, the optimization is

$$\min_{x_1,\ldots,x_T} \quad \sum_{t=1}^{T} f_t(x_t, \lambda_t) + \beta \cdot \sum_{t=1}^{T} (x_t - x_{t-1})^+ \qquad (1)$$
$$\text{subject to} \quad 0 \le x_t \in \mathbb{R}^J, \qquad x_0 = 0.$$

Note that this optimization makes two main simplifications. First, it does not impose integer constraints on x_t. This is acceptable since the number of servers is assumed to be large and so rounding does not create significant inefficiency. Second, the number of each type of server is not explicitly bounded above. An upper bound $x_t \le M_t$ can be imposed by defining $f_t(x, \cdot) = \infty$ for $x > M_t$. A constraint on the load per server can be imposed similarly. It is this formulation of the cost optimization that we focus on in the remainder of the paper.

Given this optimization problem, in many cases the solution can be found easily *offline*, i.e., given λ_t for all t. However, our goal is to find *online* algorithms for this optimization, i.e., algorithms that determine x_t using only information up to time $t + w$, where $w \ge 0$ is called the "prediction window".

In order to evaluate the performance of online algorithms we use the standard notion of *competitive ratio*. The competitive ratio of an algorithm $\mathcal{A}$ is defined as the maximum, taken over all possible inputs, of $cost(\mathcal{A})/cost(OPT)$, where $cost(\mathcal{A})$ is the objective function of (1) under algorithm $\mathcal{A}$ and OPT is the optimal offline algorithm.

III. Homogeneous systems

In this section, we consider the homogeneous systems ($S = 1$) with $f_t(x_t, \lambda_t)$ convex in x_t. We will first characterize the optimal solution, and then study the online algorithms.

A. The optimal offline solution

It turns out that there is a simple characterization of the optimal offline solution to the data center optimization problem. The optimal x^*_τ can be viewed as 'lazily' staying within two 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

$$\text{minimize} \quad \sum_{t=1}^{\tau} f_t(x_t, \lambda_t) + \beta \sum_{t=1}^{\tau} (x_t - x_{t-1})^+ \qquad (2)$$
$$\text{subject to} \quad x_t \ge 0, \qquad x_0 = 0.$$

Then, define $x^L_\tau = x^L_{\tau,\tau}$. Similarly, let $(x^U_{\tau,1}, \ldots, x^U_{\tau,\tau})$ be the solution vector to the following optimization problem

$$\text{minimize} \quad \sum_{t=1}^{\tau} f_t(x_t, \lambda_t) + \beta \sum_{t=1}^{\tau} (x_{t-1} - x_t)^+ \qquad (3)$$
$$\text{subject to} \quad x_t \ge 0, \qquad x_0 = 0.$$

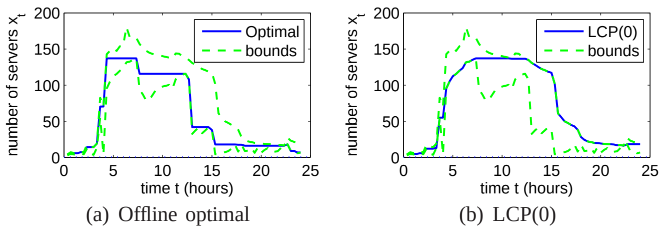


Fig. 1. Illustrations of (a) the offline optimal solution and (b) LCP(0) for the first day of the MSR workload described in Section V.

Then, define $x^U_\tau = x^U_{\tau,\tau}$.

Notice that in each case, the optimization problem includes only times $1 \le t \le \tau$, 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.

We now characterize the optimal solution x^*_τ. Define $(x)^b_a = \max(\min(x, b), a)$ as the projection of x into $[a, b]$. Then, we have:

Theorem 1. *In homogeneous system ($S = 1$) with $f_t(x_t, \lambda_t)$ convex in x_t, the optimal solution $X^* = (x^*_0, \ldots, x^*_T)$ of the data center optimization problem (1) satisfies the following backward recurrence relation*

$$x^*_\tau = \begin{cases} 0, & \tau \ge T; \\ (x^*_{\tau+1})^{x^U_\tau}_{x^L_\tau}, & \tau \le T-1. \end{cases}$$

Theorem 1 is proven in [3]. An example of the optimal x^*_t can be seen in Figure 1(a). 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.

B. Online Algorithms

Let us first present an online algorithm, *Lazy Capacity Provisioning* (LCP(w)), which is motivated by the structure of the optimal offline solution described in Section III-A. At time τ, LCP(w) knows only λ_t for $t \le \tau + w$, for some prediction window w. Like the optimal solution, it "lazily" stays within upper and lower bounds. However, it does this moving forwards in time instead of backwards in time.

To use the knowledge of the prediction window, Let us define refined bounds $x^{U,w}_\tau$ and $x^{L,w}_\tau$ such that $x^{U,w}_\tau = x^U_{\tau+w,\tau}$ in the solution of (3) and $x^{L,w}_\tau = x^L_{\tau+w,\tau}$ in that of (2). Note that $x^{U,0}_\tau = x^U_\tau$ and $x^{L,0}_\tau = x^L_\tau$. Then we are ready to define LCP(w) using $x^{U,w}_\tau$ and $x^{L,w}_\tau$.

Algorithm 1. ***Lazy Capacity Provisioning, LCP(w).***
Let $X^{LCP(w)} = (x^{LCP(w)}_0, \ldots, x^{LCP(w)}_T)$ denote the vector of active servers under LCP(w). This vector can be calculated using the following forward recurrence relation

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

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. However, it is possible to calculate

$x_{\tau}^{U,w}$ and $x_{\tau}^{L,w}$ without using the full history and hence LCP(w), remains tractable even as τ grows. Please see [3] for the details.

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. We have the following theorem:

Theorem 2. *In homogeneous system ($S = 1$) with $f_t(x_t, \lambda_t)$ convex in x_t, $cost(X^{LCP(w)}) \leq cost(X^*) + 2cost_{switching}(X^*)$. Thus, LCP(w) is 3-competitive for optimization (1). 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.*

Theorem 2 is proven in [3]. Note 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 $f_t(\cdot)$. Surprisingly, this means that even the "myopic" LCP(0) is 3-competitive. Moreover, the fact that $cost(X^{LCP(w)}) \leq cost(X^*) + 2cost_{switching}(X^*)$ highlights that the gap will tend to be much smaller in practice. However, it is a little disappointing that even for large w, the competitive ratio of $LCP(w)$ is arbitrarily close to 3.

Next, let us study the traditional *Receding Horizon Control* (RHC), which turns out to have the competitive ratio decreases as w increases. RHC is commonly proposed for dynamic capacity provisioning [4], [5] and has a long history in the control theory literature [7]–[9]. Informally, RHC works by, at time τ, solving the cost optimization over the window $(\tau, \tau + w)$ given the starting state $x_{\tau-1}$. Formally, define $X^{\tau}(x_{\tau-1}; \lambda)$ as the vector in $(\mathbb{R}^J)^{w+1}$ indexed by $t \in \{\tau, \ldots, \tau + w\}$, which is the solution to

$$\min_{x_\tau, \ldots, x_{\tau+w}} \sum_{t=\tau}^{\tau+w} f_t(x_t, \lambda_t) + \beta \cdot \sum_{t=\tau}^{\tau+w} (x_t - x_{t-1})^+ \qquad (4)$$
$$\text{subject to} \quad x_t \geq 0$$

Then, RHC works as follows.

Algorithm 2. ***Receding Horizon Control, RHC.***
For all $t \leq 0$, set the number of active servers to $x_{RHC,t} = 0$. At each timeslot $\tau \geq 1$, set the number of active servers to

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

and optimally dispatch λ_τ across $x_{RHC,\tau}$.

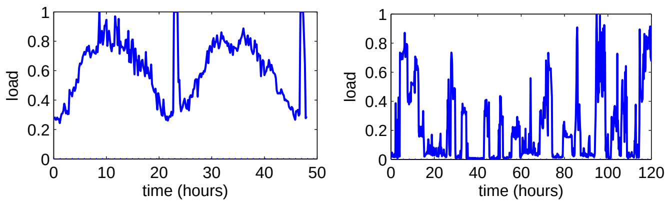
Note that (4) need not have a unique solution. We define RHC to select the solution with the greatest first entry. Define f_0 the minimum cost per timeslot for an active server, then we have the following theorem:

Theorem 3. *In homogeneous system ($S = 1$) with $f_t(x_t, \lambda_t)$ convex in x_t, RHC is $(1 + \frac{\beta}{(w+1)f_0})$-competitive.*

Theorem 3 is proven in [6]. It highlights that, with enough lookahead, RHC is guaranteed to perform quite well in the homogeneous case. However, RHC can have bad performance if w is small and β is large.

IV. Heterogeneous systems

Unfortunately, the story is different when servers are *heterogenous* ($S \geq 2$). In a heterogeneous system, x_t is a vector and it seems hard to extend $LCP(w)$ to the heterogeneous



(a) Hotmail (b) MSR

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

case. Moreover, RHC may not see any improvement in the competitive ratio as w increases. The following theorem is proven in [6]:

Theorem 4. *When there are multiple types of servers ($S \geq 2$), for all $w \geq 0$ RHC is $\geq (1 + \max_s(\beta_s/f_{0,s}))$-competitive.*

Further, the worst case instance used to prove Theorem 4 uses convex f, so the hardness is truly coming from the heterogeneity and not from other factors. To addresses the limitations of RHC in the heterogeneous setting, we propose the following novel variant.

First consider a family of algorithms parameterized by $k \in [1, w+1]$ that recompute their provisioning periodically. For all $k = 1, \ldots, w+1$, let $\Omega_k = \{i : i \equiv k \mod (w+1)\} \cap [-w, \infty)$;

Algorithm 3. ***Fixed Horizon Control, version k, FHC$^{(k)}$.***
For all $t \leq 0$, set the number of active servers to $x_{FHC,t}^{(k)} = 0$. At each timeslot $\tau \in \Omega_k$, for all $t \in \{\tau, \ldots, \tau + w\}$, set

$$x_{FHC,t}^{(k)} = X_t^{\tau}\left(x_{FHC,\tau-1}^{(k)}; \lambda\right)$$

using (4), and dispatch λ_t over $x_{FHC,t}^{(k)}$ optimally.

The above algorithm can have very poor performance. However, it gives rise to the following useful algorithm, AFHC, which averages the decisions of the $w+1$ FHC algorithms to ensure good performance.

Algorithm 4. ***Averaging Fixed Horizon Control, AFHC.***
At timeslot $\tau \in \Omega_k$, use FHC$^{(k)}$ to determine the provisioning $x_{k,\tau}^{(k)}, \ldots, x_{k,\tau+w}^{(k)}$, and then set $x_{AFHC,t} = \sum_{k=1}^{w+1} x_{k,t}^{(k)}/(w+1)$. At time t, λ_t is dispatched optimally over $x_{AFHC,t}$.

Intuitively, AFHC seem worse than RHC because RHC uses the latest information to make the current decision and AFHC make decisions in advance, thus ignoring some possibly valuable information. This intuition is partially true, as shown in the following theorem (proven in [6]), which states that RHC is not worse than AFHC for any workload in a homogeneous system.

Theorem 5. *In homogeneous system ($S = 1$), $cost(RHC) \leq cost(AFHC)$*

However, RHC can be worse than AFHC in heterogeneous systems, even when there are only two types of servers. Moreover, the following theorem highlights that AFHC guarantees good performance in the homogeneous and the heterogeneous setting.

Theorem 6. *If $f_t(\alpha x_t, \alpha \lambda_t) = \alpha f_t(x_t, \lambda_t)$ for all $\alpha > 0$, or $f_t(x_t, \lambda_t)$ is convex in x_t, then AFHC is $(1 + \max_s \frac{\beta_s}{(w+1)f_{0,s}})$-competitive.*

The above theorem is also proven in [6]. The contrast between Theorem 4 and Theorems 6 highlights the improvement AFHC provides over RHC. In fact, AFHC has the same competitive ratio in the general (possibly heterogeneous) setting that RHC has in the homogeneous setting.

V. Case studies

In this section our goal is two-fold: First, we seek to evaluate the cost incurred by online algorithms 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 using two real-world traces. We have attempted to choose experimental settings so that the benefit of dynamic right-sizing is conservatively estimated. For simplicity, we just show some results for homogeneous system and $LCP(w)$ algorithm. More experiments can be found in [3] and [6].

A. Experimental setup

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 an optimal '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.

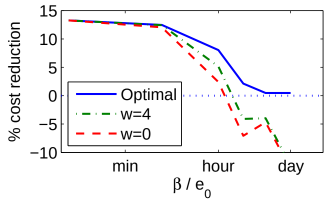
Cost function: The operating costs are a weighted sum of delay costs and energy costs with typical settings [3]. The normalized switching cost β/e_0 measures the duration a server must be powered down to outweigh the switching cost including power, wear-and-tear and so on.

Workload information: The workloads for these experiments are drawn from two real-world data center I/O traces [3]. The first set of traces is from Hotmail and the second set of traces is from MSR Cambridge. 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.

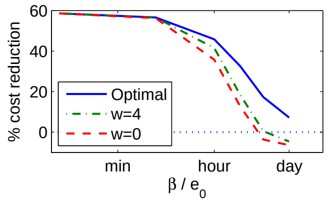
B. Impact of switching costs

One of the main worries when considering right-sizing is the switching cost of toggling servers β. Thus, an important question to address is: "How large must switching costs be before the cost savings from right-sizing disappears?"

Figure 3 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. Though the gains are smaller in the Hotmail case for large β, this is because the spike of background work splits an 8 hour valley into two short 4 hour valleys. If these tasks were shifted or balanced across the valley, the Hotmail trace would show better cost reduction.

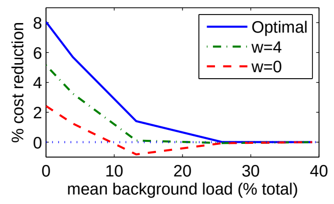


(a) Hotmail

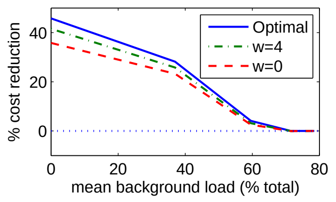


(b) MSR

Fig. 3. Impact of switching cost, against time on a logarithmic scale.



(a) Hotmail



(b) MSR

Fig. 4. 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.

C. 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 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 4 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%.

VI. Related work

Interest in right-sizing has been growing since [10] and [11] appeared at the start of the decade. Early systems work such as [11] achieved substantial savings despite ignored switching costs in their design. Other designs have focused on decentralized frameworks, e.g., [12] and [13]. A recent survey is [14]. Related analytic work focusing on dynamic right-sizing includes [15], which reallocates resources between tasks within a data center, and [16], which considers sleep of individual components, among others. Typically, approaches have applied optimization using queueing theoretic models, e.g., [17], [18], or control theoretic approaches, e.g., [19]–[21]. A recent survey of analytic work focusing on energy efficiency in general is [22]. 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.

VII. Summary and concluding remarks

This paper has discussed recent work presenting new online algorithms for dynamic right-sizing in data centers. The algorithm $LCP(w)$ 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. We also show that the classic *Receding Horizon Control* is $1 + O(1/w)$-competitive when the system is homogeneous, but when the system is heterogeneous it can perform badly — the competitive ratio does not improve as the size of the prediction window, w, grows. Accordingly, we discussed a newly proposed variant of RHC which is able to provide $1 + O(1/w)$-competitive ratio even in the heterogeneous setting.

References

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

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

[3] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *Proc. IEEE INFOCOM*, 2011, pp. 1098–1106.

[4] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," in *IEEE Int. Symp. High Performance Computer Architecture (HPCA)*, 2008, pp. 101–110.

[5] 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.

[6] M. Lin, L. Andrew, and A. Wierman, "Dynamic capacity provisioning of heterogeneous servers," Under Submission.

[7] W. Kwon and A. Pearson, "A modified quadratic cost problem and feedback stabilization of a linear system," *IEEE Trans. Automatic Control*, vol. AC-22, no. 5, pp. 838–842, 1977.

[8] W. H. Kwon, A. M. Bruckstein, and T. Kailath, "Stabilizing state feedback design via the moving horizon method." *Int. J. Contr.*, vol. 37, no. 3, pp. 631–643, 1983.

[9] D. Q. Mayne and H. Michalska, "Receding horizon control of nonlinear systems," *IEEE Trans. Automat. Contr.*, vol. 35, no. 7, pp. 814–824, 1990.

[10] 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 Symp. Operating System Principles (SOSP)*, 2001, pp. 103–116.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

[18] 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.

[19] T. Horvath and K. Skadron, "Multi-mode energy management for multi-tier server clusters," in *Proc. ACM Int. Conf. Parallel Architectures and Compilation Techniques (PACT)*, 2008, p. 1.

[20] 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.

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

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