Data Centers Power Reduction: A two Time Scale Approach for Delay Tolerant Workloads

Yuan Yao*, Longbo Huang*, Abhihshek Sharma*, Leana Golubchik* and Michael Neely*
*University of Southern California, Los Angeles, CA 90089
Email: {yuanyao, longbohu, absharma, leana, mjneely}@usc.edu

***Abstract*—In this work we focus on a stochastic optimization based approach to make distributed routing and server management decisions in the context of large-scale, geographically distributed data centers, which offers significant potential for exploring power cost reductions. Our approach considers such decisions at different time scales and offers provable power cost and delay characteristics. The utility of our approach and its robustness are also illustrated through simulation-based experiments under delay tolerant workloads.**

I. Introduction

Over the last few years, the demand for computing has grown significantly. This demand is being satisfied by very large scale, geographically distributed data centers, each containing a huge number of servers. While the benefits of having such infrastructure are significant, so are the corresponding energy costs. As per the latest reports, several companies own a number of data centers in different locations, each containing thousands of servers – Google (≈1 million), Microsoft (≈200K), Akamai (60-70K), INTEL (≈100K), and their corresponding power costs are on the order of millions of dollars per year [1]. Given this, a reduction by even a few percent in power cost can result in savings of millions of dollars.

Extensive research has been carried out to reduce power cost in data centers, e.g., [2], [3], [4], [5], [6], [7]; such efforts can (in general) be divided into the following two categories. Approaches in the first category attempt to save power cost through power efficient hardware design and engineering, which includes designing energy efficient chips, DC power supplies, and cooling systems. Approaches in the second category exploit three different levels of power cost reduction at existing data centers as follows. Firstly, at the server level, power cost reduction can be achieved via power-speed scaling [5], where the idea is to save power usage by adjusting the CPU speed of a single server. Secondly, at the data center level, power cost reduction can be achieved through data center right sizing [7], [4], where the idea is to dynamically control the number of activated servers in a data center to save power. Thirdly, at the inter-data center level, power cost reductions can be achieved by balancing workload across centers [2], [3], where the idea is to exploit the price diversity of geographically distributed data centers and route more workload to places where the power prices are lower.

This material is supported in part by the NSF Career grant CCF-0747525.

Our work falls under the second category, where our goal is to provide a unifying framework that allows one to exploit power cost reduction opportunities across all these levels. Moreover, the *non-work-conserving* nature of our framework allows us to take advantage of the temporal volatility of power prices while offering an *explicit* tradeoff between power cost and delay.

Consider a system of M geographically distributed data centers, each consisting of a front end proxy server and a back end server cluster as shown in Figure 1. At different time instances, workload arrives at the front end proxy servers which have the flexibility to distribute this workload to different back end clusters. The back end clusters receive the workload from front end servers and have the flexibility to choose when to serve that workload by managing the number of activated servers and the service rate of each server.

The problem then is to make the following three decisions, with the objective of reducing power cost: (i) how to distribute the workload from the front end servers to the back end clusters, (ii) how many servers to activate at each back end cluster at any given time, and (iii) how to set the service rates (or CPU power levels) of the back end servers.

Our proposed solution exploits temporal and spatial variations in the workload arrival process (at the front end servers) and the power prices (at the back end clusters) to reduce power cost. It also facilitates a *cost vs. delay* trade-off which allows data center operators to reduce power cost at the expense of increased service delay. Hence, our work is suited for *delay tolerant workloads* such as massively parallel and data intensive MapReduce jobs. Today, MapReduce programming based applications are used to build a wide array of web services – e.g., search, data analytics, social networking, etc. Hence, even though our proposed solution is more effective for delay tolerant workloads it is still relevant to many current and future cloud computing scenarios.

Our contributions can be summarized as follows:

- We propose a *two time scale* control algorithm aimed at reducing power cost and facilitating a power cost vs. delay trade-off in geographically distributed data centers (Section II and III).
- By extending the traditional Lyapunov optimization approach, which operates on a single time scale, to two different time scales, we derive analytical bounds on the time average power cost and service delay achieved by our

algorithm (Section IV).

- Through simulations based on real-world data sets, we show that our approach can reduce the power cost by as much as 18%, even for state-of-the-art server power consumption profiles and data center designs (Section VI). We also show that our approach is environmentally friendly, in the sense that it not only reduces power cost but also the actual power usage.
- We demonstrate the robustness of our approach to errors in estimating data center workload both analytically as well as through simulations (Sections VI-C and VI-D).

II. Problem Formulation

We first formulate our problem and then discuss the practical aspects of our model. We consider M geographically distributed data centers, denoted by $\mathcal{D} = \{D_1, ..., D_M\}$, where the system operates in slotted time, i.e., $t = 0, 1, 2, ...$ Each data center D_i consists of two components, a front end proxy server, S_i^F, and a back end server cluster, $\mathcal{S}_i^B$, that has N_i homogeneous servers. Fig. 1 depicts our system model. Below we first present our model's components.

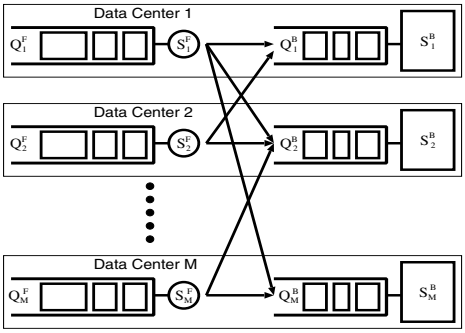

Fig. 1. A model of M geographically distributed data centers.

A. The Workload Model

In every time slot t, jobs arrive at each data center. We denote the amount of workload arriving at D_i by $A_i(t)$, where $\boldsymbol{A}(t) = (A_1(t), ..., A_M(t))$ denotes the arrival vector. In our analysis, we first assume that $\boldsymbol{A}(t)$ are i.i.d. every time slot with $\mathbb{E}\{\boldsymbol{A}(t)\} = \boldsymbol{\lambda} \triangleq (\lambda_1, ..., \lambda_M)$. We later discuss how our results can be extended to the case when $\boldsymbol{A}(t)$ evolve according to more general random processes. We also assume that there exists some A_{max} such that $A_i(t) \leq A_{max}$ for all i and t. Note that the components in $\boldsymbol{A}(t)$ can be correlated – this is important in practice as arrivals to different data centers may be correlated.

B. The Job Distribution and Server Operation Model

A job first arrives at the front end proxy server of D_i, S_i^F, and is queued there. The proxy server S_i^F then decides how to distribute the awaiting jobs to the back end clusters at different data centers for processing. To model this decision, we use $\mu_{ij}(t)$ to denote the amount of workload routed from D_i to D_j at time t, and use $\boldsymbol{\mu}_i(t) = (\mu_{i1}(t), ..., \mu_{iM}(t))$ to denote the vector of workload routing rates at S_i^F. We assume that in every t, $\boldsymbol{\mu}_i(t)$ must be drawn from some general *feasible* workload distribution rate set $\mathcal{R}_i$, i.e., $\boldsymbol{\mu}_i(t) \in \mathcal{R}_i$ for all t. We assume that each set $\mathcal{R}_i$ is time invariant and compact. It can be either continuous or finite discrete. Also each set $\mathcal{R}_i$ contains the constraint that $\sum_j \mu_{ij}(t) \leq \mu_{max}$ for some finite constant μ_{max}. Note that this assumption is quite general. For example, $\mathcal{R}_i$ can contain the constraints that $\mu_{ij}(t) = 0$ for all t to represent the constraint that jobs arriving at D_i cannot be processed at D_j, e.g., due to the fact that D_j does not have a data set needed for the corresponding computation.

For each data center D_i, the jobs routed to its back end cluster are queued in a *shared* buffer. The data center then controls its back end cluster as follows. In every time slot of the form $t = kT$ with $k = 0, 1, ...$ where $T \geq 1$, the data center first decides on the number of servers to activate. We denote the number of active servers at time t at D_i as $N_i(t)$, where $N_i(t) \in \{N_{min}^i, N_{min}^i + 1, N_{min}^i + 2, ..., N_i\}$, with N_i being the total number of servers at the back end cluster S_i^B, and $N_{min}^i, 0 \leq N_{min}^i \leq N_i$ being minimum number of servers that should be activated at all times for data center D_i. If at time slot $t = kT$, we have $N_i(t) > N_i(t - T)$, then we activate more servers. Otherwise, we simply put $N_i(t - T) - N_i(t)$ servers to sleep. The reasons for having $N_i(t)$ changed only every T time slots are: (i) activating servers typically costs a non-negligible amount of time and power, and (ii) frequently switching back and forth between active and sleep states can result in reliability problems [8]. In addition to deciding on the number of active servers, the data center sets the service rate of each active server every time slot. This can be achieved by using techniques such as power scaling [5]. For ease of presentation, below we assume that all active servers in a data center operate at the same service rate, and denote active servers' service rate at D_i by $b_i(t), 0 \leq b_i(t) \leq b_{max}$, where b_{max} is some finite number. We note that our model can be extended to more general scenarios. A further discussion of above assumptions is given in Section II-E.

C. The Cost Model

We now specify the cost model. For time slot t, we use $t_T = \lfloor \frac{t}{T} \rfloor T$ to denote the last time before t when the number of servers has been changed. Then at time slot t, by running $N_i(t_T)$ servers at speed $b_i(t)$, D_i consumes a total power of $P_i(N_i(t_T), b_i(t))$. We assume that the function $P_i(\cdot, \cdot)$ is known to D_i, and there exists some P_{max} such that $P_i(N_i(t_T), b_i(t)) \leq P_{max}$ for all t and i. Such power consumption will in turn incur some monetary cost for the data centers, of the form "power×price". To also model the fact that each data center may face a different power price at time slot t, we denote the power price at D_i at time slot t by $p_i(t)$. We assume that $\boldsymbol{p}(t) = (p_1(t), ..., p_M(t))$ varies every $T_1 \geq 1$ time slots, where $T = cT_1$ for some integer c. We assume $\boldsymbol{p}(t)$ are i.i.d and every T_1 time slot, each $p_i(t)$ takes a value in some finite state space $\mathcal{P}_i = \{p_i^1, ..., p_i^{|\mathcal{P}_i|}\}$. We also define $p_{max} \triangleq \max_{ij} p_i^j$ as the maximum power price that any data center can experience. We use $\pi_i(p)$ to denote the marginal probability that $p_i = p$. An example of these different time scales is given in Figure 2.

Finally we use $f_i(t) = P_i(N_i(t_T), b_i(t))p_i(t)$ to denote the power cost at D_i in time slot t. It is easy to see that if we define $f_{max} \triangleq MP_{max}p_{max}$, then $\sum_i f_i(t) \leq f_{max}$ for all t.

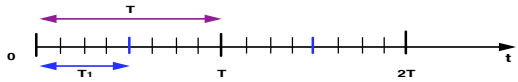

Fig. 2. An example of different time scales T and T_1. In this example, $T = 8$, $T_1 = 4$, and $T = 2T_1$.

D. The data center Power Cost Minimization (PCM) problem

Let $\boldsymbol{Q}(t) = (Q_i^F(t), Q_i^B(t), i = 1, ..., M)$, $t = 0, 1, ...$, be the vector denoting the workload queued at the front end servers and the back end clusters at time slot t. We use the following queueing dynamics:

$$Q_i^F(t+1) = \max\left[Q_i^F(t) - \textstyle\sum_j \mu_{ij}(t), 0\right] + A_i(t), \quad (1)$$

$$Q_i^B(t+1) \leq \max\left[Q_i^B(t) - N_i(t)b_i(t), 0\right] + \textstyle\sum_j \mu_{ji}(t). \quad (2)$$

The inequality in (2) is due to the fact that the front end servers may allocate a total routing rate that is more than the actual workload queued. In the following, we assume that data centers can estimate the unfinished workload in their queues accurately. The case when such estimation has errors will be discussed in Section IV. Throughout the paper, we use the following definition of queue stability:

$$\overline{Q} \triangleq \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \sum_{i=1}^{M} \mathbb{E}\{Q_i^F(\tau) + Q_i^B(\tau)\} < \infty. \quad (3)$$

Then we define a *feasible* policy to be the one that chooses $N_i(t)$ every T time slots subject to $N_{min}^i \leq N_i(t) \leq N_i$, and chooses $\mu_{ij}(t)$ and $b_i(t)$ every time slot subject to only $\boldsymbol{\mu}_i(t) \in \mathcal{R}_i$ and $0 \leq b_i(t) \leq b_{max}$. We then define the time average cost of a policy Π to be:

$$f_{av}^{\Pi} \triangleq \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \sum_{i=1}^{M} \mathbb{E}\{f_i^{\Pi}(\tau)\}. \quad (4)$$

Here, $f_i^{\Pi}(\tau)$ denotes the power cost incurred by policy Π at time slot τ. We call every feasible policy that ensures (3) a *stable* policy, and use f_{av}^* to denote the infimum average power cost over all stable policies. The objective of our problem is to find a stable policy that chooses the number of activated servers $N_i(t)$ every T time slots, and chooses the workload distribution rates $\mu_{ij}(t)$ and the service rates $b_i(t)$ every single time slot, so as to minimize the time average power cost. We refer to this as the data center Power Cost Minimization (PCM) problem in the remainder of the paper.

E. Model Discussion and Practical Consideration

We now discuss the assumptions made in our model.

Above, we assume that in time slot t all activated servers at D_i have the same service rate $b_i(t)$. This is not restrictive. Indeed, let us focus on one data center in a single time slot and consider the following formulation: Let the power consumption of a server with service rate b be $P_{server}(b)$, a convex function of b (see [6], [9]). If the N activated servers run at service rates b_1, b_2, ... , b_N, then the total power consumed by them can be written as $P_{total} = \sum_{j=1}^{N} P_{server}(b_j)$. Suppose the actual power consumption in this data center, P_{center}, has the form $P_{center} = C_1 \times P_{total} + C_2$, where C_1 and C_2 are constants or functions of N only. Then we have:

$$P_{center} = C_1 P_{total} + C_2 = C_1 \sum_{i=1}^{N} P_{server}(b_i) + C_2$$

$$\geq C_1 N P_{server}\left(\frac{\sum_{i=1}^{N} b_i}{N}\right) + C_2.$$

This indicates that, to minimize the power consumption without reducing the amount of workload served, all servers should have the same service rate. This justifies our assumption.

We also assume that jobs can be served at more than one data center. When serving certain types of jobs, such as I/O intensive ones, practical considerations such as data locality should also be considered. This scenario can easily be accommodated in our model by imposing restrictions on $\mathcal{R}_i$ at the front end servers. Moreover, service providers like Google replicate data across (at least) two data centers [10]. This provides flexibility for serving I/O intensive jobs.

We also assume that the data centers can observe/measure $\mathbf{Q}(t)$, i.e., the unfinished work of all incoming workload accurately. However, in many cases, the available information only contains the number of jobs and the number of tasks per job. With this, we can estimate the amount of workload of the incoming jobs. In addition, in Section IV we show that even if the estimation is not accurate we can still prove bounds on the performance of our approach. Moreover, in Section VI-C we show the robustness of our approach against workload estimation errors through experiments.

Finally, we assume that a server in sleep state consumes much less power than an idle server. According to [7], a server consumes 10 Watts when in sleep state, as compared to 150 Watts in idle state. This indicates that our assumption is reasonable. We also assume that servers can be activated and put to sleep immediately. We note that waking up from sleep takes around 60 seconds. During this 60 seconds, the server cannot perform any work. This should not be ignored, if the control actions on activating servers are made frequently. However, when we choose T, the period of such actions, to be large, potentially no less than an hour, we can assume that the wake up time is amortized over the relatively long period during which the server is active. The effect of T is further discussed in Section VI, where we give experimental results.

III. The SAVE algorithm

We solve PCM through our StochAstic power redUction schEme (SAVE). We first describe SAVE's control algorithms and discuss corresponding insight and implementation related issues. SAVE's derivation and analytical performance bounds are discussed in the following section.

A. SAVE's control algorithms

SAVE's three control actions are:

- Front end Routing: In every time $t = kT$, $k = 0, 1, ...$, each D_i solves for μ_{ij} to maximize

$$\sum_{j=1}^{M} \mu_{ij}[Q_i^F(t) - Q_j^B(t)] \quad (5)$$

subject to the constraint that $\boldsymbol{\mu}_i = (\mu_{i1}, ..., \mu_{iM}) \in \mathcal{R}_i$. Then in every time slot $\tau \in [t, t+T-1]$, D_i distributes up to $\mu_{ij}(\tau)$ amount of workload to the back end cluster at D_j, $1 \leq j \leq M$.

- Back end Server Management: At time slot $t = kT$, data center D_i chooses the number of servers $N_i(t) \in [N_{min}^i, N_i]$ to minimize:

$$\mathbb{E}\Big\{ \sum_{\tau=t}^{t+T-1} \sum_j \big[V f_j(\tau) - Q_j^B(t) N_j(t) b_j(\tau)\big] | \boldsymbol{Q}(t) \Big\} \quad (6)$$

Then data center D_i uses $N_i(t)$ servers over time $[t, t+T-1]$. In every time slot $\tau \in [t, t+T-1]$, each data center D_i chooses the service rate of the servers $b_i(\tau)$ (note that $N_i(t)$ is determined at time slot t) to minimize:

$$V f_j(\tau) - Q_j^B(t) N_j(t) b_j(\tau) \quad (7)$$

- Queue Update: Update the queues using (1) and (2).

Note that $V > 0$ is a parameter of SAVE that controls the tradeoff between power cost and delay performance, as detailed below in Theorem 2. SAVE works at two different time scales. The front end routing decisions and number of active servers selection, $N_i(t)$, are made every T slots, while back end servers' service rates are updated at each slot. This two time scale mechanism is important from an implementation perspective because waking up servers from sleep state usually takes much longer than servers' speedscaling. The *Back end Server Management* step involves maximizing (6), an expectation over future (power cost) events. We show in Section III-C that this can be carried out through learning.

B. Properties of SAVE

We highlight SAVE's two interesting properties. First, it is not work-conserving. A back end cluster $\mathcal{S}_j^B$ may choose not to serve jobs in a particular time slot, even if $Q_j^B > 0$, due to a high power price at D_j. This may introduce additional delay but can reduce the power cost as shown in Section VI-A.

Second, SAVE can provide opportunities for bandwidth cost savings because (a) it provides an explicit upper bound on the workload sent from S_i^F to $\mathcal{S}_j^B$, and (b) these routing decisions remain unchanged for T time slots. If T is large, this can provide opportunities for data centers to optimize network routing ahead of time to reduce bandwidth cost. As highlighted in [2], content distribution networks like Akamai can incur significant bandwidth costs. Incorporating bandwidth costs into our model is part of future work.

C. Implementing the algorithm

Note that the routing decisions made at the front end servers do not require any statistical knowledge of the random arrivals and prices. All that is needed is that D_i's back end cluster broadcasts $Q_i^B(t)$ to all front end proxy servers every T time units. This typically only requires a few bits and takes very little time to transmit. Then each data center D_i compute μ_{ij} for each j. The complexity of maximizing $\sum_j \mu_{ij}[Q_i^F(t) - Q_j^B(t)]$ depends on the structure of the set $\mathcal{R}_i$. For example, if $\mathcal{R}_i$ only allows one μ_{ij} to take nonzero values, then we can easily find an optimal solution by comparing the weight-rate-product of each link, and finding the best one.

In the second step, we need to minimize (6). This in general requires statistical knowledge of the power prices. And it is obtained as follows: At every $t = kT$, $k \geq 0$, we use the empirical distribution of prices over a time window of size L to compute the expectation. Specifically, if $t \geq L$, then let $n_p^i(t, L)$ be the number of times the event $\{p_i(t) = p\}$ appears in time interval $[t-L+1, t]$. Then use $\frac{n_p^i(t,L)}{L}$ as the probability $\pi_i(p)$ for estimating the expectations. Since all $p_i(t)$ only take finitely many values, we have:

$$\lim_{L\to\infty} \frac{n_p^i(t, L)}{L} = \pi_i(p). \quad (8)$$

Therefore, as we increase the number of samples, the estimation becomes better and better. Note that in this procedure, we use the fact that (6) can be decomposed into a summation of M expectations, and that each expectation only requires the marginal distribution of the prices. Now we can solve for $N_j(t)$ and $b_j(\tau)$ through a joint optimization of (6). Note that this is a two variable optimimization, since in an optimal solution $b_j(\tau)$ remains constant when $\tau \in [t, t+T-1]$.

IV. SAVE: Design and Performance Analysis

SAVE's focus on reducing power cost along with queue stability suggests a design approach based on the Lyapunov optimization framework [11]. This framework allows us to include power costs into the *Lyapunov drift analysis*, a well-known technique for designing stable control algorithms. However, the *vanilla* Lyapunov optimization based algorithms operate on a single time scale. We extend this approach to two different time scales, and derive analytical performance bounds analogous to the single time scale case. We now highlight the key steps in deriving SAVE, and then characterize its power cost and delay performance.

A. Algorithm Design

We first define the Lyapunov function, $L(t)$, that measures the aggregate queue backlog in the system.

$$L(t) \triangleq \sum_{i=1}^{M} \big([Q_i^F(t)]^2 + [Q_i^B(t)]^2\big). \quad (9)$$

Next, we define the T-slot Lyapunov drift, $\Delta_T(t)$ as the expected change in the Lyapunov function over T slots.

$$\Delta_T(t) \triangleq \mathbb{E}\big\{L(t+T) - L(t) | \boldsymbol{Q}(t)\big\}. \quad (10)$$

Following the Lyapunov optimization approach, we add the expected power cost over T slots (i.e., a penalty function), $\mathbb{E}\big\{\sum_{\tau=t}^{t+T-1} \sum_j f_j(\tau)\big\}$, to (10) to obtain the *drift-plus-penalty* term. A key derivation step is to obtain an upper bound on this term. The following lemma defines such an upper bound for our case (see [12] for proof).

Lemma 1. *Let $V > 0$ and $t = kT$ for some nonnegative integer k. Then under any possible actions $N_i(t) \in [N_{min}^i, N^i]$, $\boldsymbol{\mu}_i(t) \in \mathcal{R}_i$ and $b_i(t) \in [0, b_{max}]$, we have:*

$$\Delta_T(t) + V\mathbb{E}\Big\{ \sum_{\tau=t}^{t+T-1} \sum_j f_j(\tau) | \boldsymbol{Q}(t) \Big\} \leq$$

$$B_1T + V\mathbb{E}\Big\{\sum_{\tau=t}^{t+T-1}\sum_j f_j(\tau)|\boldsymbol{Q}(t)\Big\}$$
$$-\mathbb{E}\Big\{\sum_{\tau=t}^{t+T-1}\sum_i Q_i^F(\tau)\big[\sum_j \mu_{ij}(\tau) - A_i(\tau)\big]|\boldsymbol{Q}(t)\Big\}$$
$$-\mathbb{E}\Big\{\sum_{\tau=t}^{t+T-1}\sum_j Q_j^B(\tau)\big[N_j(t)b_j(\tau) - \sum_i \mu_{ij}(\tau)\big]|\boldsymbol{Q}(t)\Big\}. \quad (11)$$

Here $B_1 \triangleq MA_{max}^2 + \sum_i N_i^2 b_{max}^2 + (M^2+M)\mu_{max}^2$.

The main design principle in Lyapunov optimization is to choose control actions that minimize the R.H.S. of (11). However, for any slot t, this requires *prior* knowledge of the future queue backlogs ($\boldsymbol{Q}(t)$) over the time frame $[t, t+T-1]$. $\boldsymbol{Q}(t)$ depends on the job arrival processes $A_i(t)$, and SAVE's decisions $\mu_{ij}(t), b_i(t)i$, and $N_i(t)$ (that depend on time varying power prices). Hence, minimizing the R.H.S. of (11) requires information about the random job arrival and power price processes. This information may not always be available. In SAVE we address this by *approximating* future queue backlog values as the current value, i.e., $Q_i^F(\tau) = Q_i^F(t)$ and $Q_j^B(\tau) = Q_j^B(t)$ for all $t < \tau \leq t+T-1$. However, the simplification forces a "loosening" of the upper bound on the drift-plus-penalty term as shown in the following lemma (see [12] for proof).

Lemma 2. *Let $t = kT$ for some nonnegaive integer k. Then under any possible actions $N_i(t), \mu_{ij}(t), b_i(t)$ that can be taken, we have:*

$$\Delta_T(t) + V\mathbb{E}\Big\{\sum_{\tau=t}^{t+T-1}\sum_j f_j(\tau)|\boldsymbol{Q}(t)\Big\} \leq$$
$$B_2T + \mathbb{E}\Big\{\sum_{\tau=t}^{t+T-1}\sum_j Q_i(t)A_i(\tau)|\boldsymbol{Q}(t)\Big\}$$
$$-\mathbb{E}\Big\{\sum_{\tau=t}^{t+T-1}\sum_j \mu_{ij}(\tau)\big[Q_i^F(t) - Q_j^B(t)\big]|\boldsymbol{Q}(t)\Big\}$$
$$+\mathbb{E}\Big\{\sum_{\tau=t}^{t+T-1}\sum_j \big[Vf_j(\tau) - Q_j^B(t)N_j(t)b_j(\tau)|\boldsymbol{Q}(t)\Big\}. \quad (12)$$

Here $B_2 \triangleq B_1 + (T-1)\sum_j [N_j^2 b_{max}^2 + (M^2+1)\mu_{max}^2]/2$.

Comparing (12) with (5), (6), and (7), we can see that SAVE chooses $N_i(t), \mu_{ij}(t), b_i(t)$ to minimize the R.H.S. of (12).

B. Performance bounds

Theorem 2 (below) provides analytical bounds on the power cost and delay performance of SAVE. To derive these bounds, we first need to characterize the optimal time average power cost f_{av}^* that can be achieved by any algorithm that stabilizes the queue. Theorem 1 (below) shows that using a *stationary randomized algorithm* we can achieve the minimum time average power cost f_{av}^* possible for a given job arrival rate vector $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_M)$ where $\lambda_i = \mathbb{E}\{A_i(t)\}$. We define stationary randomized algorithms as the class of algorithms that choose $N_i(t), \mu_{ij}(t)$, and $b_i(t)$ according to a fixed probability distribution that depends on $A_i(t)$ and $f_j(t)$ but is independent of $\boldsymbol{Q}(t)$. In Theorem 1, $\boldsymbol{\Lambda}$ denotes the *capacity region* of the system – i.e., the closure of set of rates $\boldsymbol{\lambda}$ for which there exists a joint workload assignment and computational capacity adaptation algorithm that ensures (3).

Theorem 1. *(Optimality over Stationary Randomized Policies) For any rate vector $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$, there exists a stationary randomized control policy Π_{opt} that chooses $N_i(t), i = 1, ..., M$ every T slots, and chooses $\boldsymbol{\mu}_i(t) \in \mathcal{R}_i$ and $b_i(t) \in [0, b_{max}]$ every time slot purely as functions of $p_i(t)$ and $A_i(t)$, and achieves the following for all $k = 0, 1, 2, ...$*

$$\sum_{\tau=kT}^{kT+T-1}\sum_{i=1}^{M}\mathbb{E}\big\{f^{\Pi_{opt}}(\tau)\big\} = Tf_{av}^*(\boldsymbol{\lambda}),$$
$$\sum_{\tau=kT}^{kT+T-1}\mathbb{E}\Big\{\sum_i \mu_{ij}^{\Pi_{opt}}(\tau)\Big\} = \sum_{\tau=kT}^{kT+T-1}\mathbb{E}\big\{N_j^{\Pi_{opt}}(kT)b_j^{\Pi_{opt}}(\tau)\big\},$$
$$\sum_{\tau=kT}^{kT+T-1}\mathbb{E}\big\{A_i(\tau)\big\} = \sum_{\tau=kT}^{kT+T-1}\mathbb{E}\Big\{\sum_j \mu_{ij}^{\Pi_{opt}}(\tau)\Big\}.$$

Proof: It can be proven using Caratheodory's theorem in [11]. Omitted for brevity. ■

The following theorem presents bounds on the time average power cost and queue backlogs achieved by SAVE.

Theorem 2. *(Performance of* SAVE*) Suppose there exists an $\epsilon > 0$ such that $\boldsymbol{\lambda} + 2\epsilon\mathbf{1} \in \boldsymbol{\Lambda}$, then under the* SAVE *algorithm, we have:*

$$\overline{Q}_T \triangleq \limsup_{K\to\infty}\frac{1}{K}\sum_{k=0}^{K-1}\sum_{i=1}^{M}\mathbb{E}\big\{Q_i^F(kT) + Q_i^B(kT)\big\}$$
$$\leq \frac{B_2 + Vf_{max}}{\epsilon}, \quad (13)$$
$$f_{av}^{\text{SAVE}} \triangleq \limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\sum_{i=1}^{M}\mathbb{E}\big\{f(\tau)\big\} \leq f_{av}^* + \frac{B_2}{V}. \quad (14)$$

Here f_{av}^* is the optimal cost defined in Section II and $\mathbf{1}$ denotes the vector of all 1's.

Proof: See [12]. ■

We can extend the results in Theorem 2 to Markovian arrival processes using techniques developed in [13]. We omit the details here due to space limitations.

What happens when SAVE makes its decisions based on queue backlog estimates $\hat{\boldsymbol{Q}}(t)$ that differ from the actual queue backlogs? The following theorem shows that the SAVE algorithm is robust against queue backlog estimation errors.

Theorem 3. *(Robustness of* SAVE*) Suppose there exists an $\epsilon > 0$ such that $\boldsymbol{\lambda} + 2\epsilon\mathbf{1} \in \boldsymbol{\Lambda}$. Also suppose there exists a constant c_e, such that at all time t, the estimated backlog sizes $\hat{Q}_i^F(t)$, $\hat{Q}_i^B(t)$ and the actual backlog sizes $Q_i^F(t)$, $Q_i^B(t)$ satisfy $|\hat{Q}_i^F(t) - Q_i^F(t)| \leq c_e$ and $|\hat{Q}_i^B(t) - Q_i^B(t)| \leq c_e$. Then under the* SAVE *algorithm, we have:*

$$\overline{Q}_T \triangleq \limsup_{K\to\infty}\frac{1}{K}\sum_{k=0}^{K-1}\sum_{i=1}^{M}\mathbb{E}\big\{Q_i^F(kT) + Q_i^B(kT)\big\}$$

$$\leq \frac{B_3 + V f_{max}}{\epsilon}, \quad (15)$$

$$f_{av}^{\text{SAVE}} \triangleq \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \sum_{i=1}^{M} \mathbb{E}\{f(\tau)\} \leq f_{av}^{*} + \frac{B_3}{V}. \quad (16)$$

Here f_{av}^{*} is the optimal cost defined in Section II, and $B_3 = B_2 + 2Tc_e(\mu_{max} + A_{max} + N_{max}b_{max} + M\mu_{max})$.

Proof: See [12]. ■

By comparing the inequalities (16) and (14), we can see that with inaccurate information, we need to set V to a larger value to obtain the same time average power cost as with accurate information. However, this will result in higher average queue backlogs (compare inequalities (15) and (13)). Hence, SAVE works even with inaccurate queue backlog information but its robustness is achieved at the expense of a power cost vs. delay trafe-off. We further demonstrate SAVE's robustness using simulations in Section VI.

V. Experimental Setup

The goal of this experimental study is to evaluate SAVE under real world settings using real world data sets. Our evaluation scenario consists of 7 data centers at different geographic locations. Next, we describe the three main components of our simulations – the electricity prices and job arrivals at different data centers, the system parameters, and alternate techniques against which we compare SAVE.

A. Data sets

Electricity prices. We downloaded the hourly electricity prices for 7 hubs at different geographic locations from [14]. These hubs supply electricity to large cities such as Boston and New York, and sites like Palo Alto, CA and Ashburn, VA that host Google's data centers. To fully exploit the cost savings due to temporal power price variations, we would have preferred to have prices at a time granularity that exhibits high variability, e..g. 5 minute intervals [2]. However, since we had access to only the hourly prices, we use interpolation to generate prices at 5 minute intervals. For more details on this, please see [12].

Workload. We chose MapReduce jobs as representative of delay tolerant workloads, and generate workload according to the published statistics on MapReduce usage at Facebook [15].

Each job consists of a set of independent tasks that can be executed in parallel. A job is completed when all its tasks are completed. We make the following assumptions in our experiments: (i) all tasks belonging to a job have the same processing time; tasks from different jobs can have different processing times; (ii) jobs can be served in any of the 7 data centers; and (iii) all the tasks from the same job must be served at the same back end cluster. Regarding (i), in practice, tasks from the same MapReduce job (and other parallel computations), exhibit variability in processing times. However, techniques exist for reducing both the prevalence and the magnitude of task duration variability for MapReduce jobs [16]. Hence, (i) is not a significant oversimplification. As explained in Section II-E, we believe (ii) is also reasonable. Assumption (iii) is not required by our approach. Rather, it is motivated by the fact that, in practice, partitioning tasks from the same MapReduce job across geographically distant clusters can degrade overall performance due to network delays.

We choose the execution time of a task belonging to a particular job to be uniformly distributed between 1 and 60 seconds [1] with the "job size" distribution (i.e., number of tasks per job) given in Table V-A; these distributions (job execution time and "job size") correspond to data reported in [15].

TABLE I
Distribution of job sizes

# Tasks	1	2	10	50	100	200	400	800	4800
%	38	16	14	8	6	6	4	4	4

We generated 5 groups of delay tolerant workloads. Each group consists of 7 different *Poisson* job arrival traces – one for each cluster. Group 1 has "homogeneous" arrival rates – the arrival process for each cluster is Poisson with a rate of 15 jobs per time slot. The length of one time slot is 15 seconds. Groups 2 to 5 have "heterogeneous" arrival rates. The average arrival rate across all data centers is kept at 15 jobs per time slot. But as the index grows larger, the variance of arrival rates grows larger. For example, Group 2 has arrival rates ranging from 14 to 16 jobs every time slot, whereas Group 5 has arrival rates ranging from 12 to 18 jobs per time slot. We note that the assumption of Poisson distributed arrivals is not groundless. In fact, it is suggested by the measurements in [15].

B. Experimental Settings

Power Cost Function. SAVE can handle a wide range of power cost functions including non-convex functions. In our experiments, we model power consumption $P(N_i, b_i)$ as:

$$P(N_i(t), b_i(t)) = \left(N_i(t) \left(\frac{b_i(t)^{\alpha}}{A} + P_{idle} \right) \right) \cdot PUE \quad (17)$$

In (17), A, P_{idle}, and α are constants determined by the data center. Specifically, P_{idle} is the average idle power consumption of a server, and $b_i(t)^{\alpha}/A + P_{idle}$ gives the power consumption of a server running at rate $b_i(t)$. In our experiments we choose $\alpha = 3$, $P_{idle} = 150$ Watts, and A such that the peak power consumption of a server is 250 Watts. The model (17) and all its parameters are based on the measurements reported in [9]. The PUE term accounts for additional power usage (such as cooling) for having $N_i(t)$ servers active. According to [17] PUE values for many of today's data centers lie between 1.3 and 2. We chose $PUE = 1.3$ in all of our experiments. This choice is pessimistic, in a sense that SAVE will achieve larger power cost reductions when PUE is higher.

System Parameters. We set N_i, the number of servers in data center i, to be 1000 for all 7 data centers. Each server can serve up to 10 tasks at the same time. With a 15 jobs per slot arrival rate, this gives an average server utility of about 60%. We set the bandwidth between the front end servers and the back end clusters to a large value, based on the real world practice [10]. Hence, a large amount of workload can be sent from one data center to another within one time slot. We vary

[1] Recall that all tasks within a job have the same execution time.

the N^i_{min} across a range of values to explore its impact on SAVE's performance (see Section VI-B).

C. Simulated Schemes for Comparison

We compare SAVE the following work-conserving schemes that either represent the current practices in data center management or are simple heuristic based approaches proposed by others.

Local Computation. All the requests arriving at S^F_i are routed to $\mathcal{S}^B_i$ (the local back end); i.e., $\mu_{ii} = A_i$ and $\mu_{ij} = 0$ if $j \neq i$.

Load Balancing. The amount of workload routed from D_i to D_j, μ_{ij}, is proportional to the service capacity of D_j, regardless of D_j's power prices. Intuitively, this scheme should have good delay characteristics.

Low Price. This scheme is similar to the heuristic proposed in [2] that routes more jobs to data centers with lower power prices. However, no data center receives workload over 95th percentile of its service capacity. Due to the discreteness of job sizes and the constraint that all tasks of one job should be served at the same back end cluster, it is difficult to ensure that the cluster with the lowest power price always runs close to, but not over, its capacity. Thus, in this scheme, the workload is routed such that the long term arrival rate at the back end cluster with the lowest average power price is close to the 95th percentile of its capacity. We then route the workload to the second lowest price cluster, and so on.

In all these schemes, we assume that all servers are activated at all times.[2] However, we assume that the service rates of the servers can be tuned in every slot. We also simulate the following scheme that power downs idle servers:

Instant On/Off. Here, the routing decisions between front end servers and back end clusters are exactly the same as in the Load Balancing scheme. However, now not all servers are active in all time slots. In every slot, each data center is able to activate/put to sleep any number of servers with *no* delay or power cost, and also adjust servers' service rates. This *idealized* scheme represents the upper bound of power cost reductions achievable for the single data center case by any work-conserving scheme in our experimental settings. It is highly impractical because it assumes that servers can switch between active state and sleep state at the same frequency as power scaling (once every 15 seconds in our simulations).

VI. Experimental Results

We now evaluate SAVE through simulation based experiments, using the experimental setup described above.

A. Performance Evaluation

The performance of SAVE depends on parameters V and T. We show experimental results of all schemes on all data sets under different V and T values (with the other parameters fixed). For power cost reduction results, we use the Local Computation scheme as a baseline. For all other schemes, we show the percentage of average power cost reduction as compared to the Local Computation scheme. Specifically, let PC_X denote the average power cost of scheme X. We use $\frac{PC_{L.C.}-PC_X}{PC_{L.C.}} \times 100$ to quantify the power cost reduction due to scheme X. (Here $L.C.$ is short for Local Computation.) For delay results, we show the schemes' average delay (in number of time slots). We omit the delay results of the On/Off scheme as they are nearly identical to those of the Load Balancing scheme — the maximum difference is ≈ 0.03 time slots (0.45 second).For all comparison schemes, we show average values (power cost reduction and delay) over all arrival data sets. For SAVE, we use curves to represent average values and bars to show the corresponding ranges.

We first fix T to be 240 time slots (one hour) and run experiments with different V values. The results are shown in Figure 3(a) and (b). From Figure 3(a) we can see that as V goes from 0.01 to 100, the power cost reduction grows from an average of around 0.1% to about 18%. The On/Off scheme achieves power cost reduction of about 9.7%. If we choose V to be greater than 5 then SAVE results in larger power cost reductions than scheme On/Off. Because (i) our approach considers differences in power prices across different data centers, and (ii) our approach is not work conserving and can adjust service rates at different data centers according to power prices. We also note that the scheme Low Price gives a small power cost reduction (of 0.5%) – i.e., sending more workload to data centers with low power prices in a greedy fashion does not lead to significant savings in power cost. In Figure 3(b), we observe that when V is small (< 0.1) the average delay of SAVE is quite small and close to the delay of scheme Load Balancing. Increasing V results in larger delay as well as larger power cost reductions. In general, V in SAVE controls the trade-off between delay and power cost; e.g., when V is large, SAVE outperforms scheme On/Off (which is impractical scheme, as noted above), in power cost reduction.

We fix V to be 10 and vary T from 30 time slots (7.5 minutes) to 1080 time slots (4.5 hours), which is a sufficient range for exploring the characteristics of SAVE. (Note that servers are activated and put to sleep every 10 minutes in [4] and every hour in [19].) Corresponding results of the different schemes are shown in Figures 3(c) and (d). From Figure 3(c) we can see that changing T has relatively small effect on power cost reductions of our SAVE approach. The average power cost reduction fluctuates between 8.7% and 13.6% when T varies from 30 to 1080 time slots. In most cases, it results in higher cost reductions than scheme On/Off. However, we note that T has a larger impact on average delay, as shown in Figure 3(d). In the extreme case, when $T = 1080$ time slots, the average delay is close to 64 time slots. This is not surprising – recall that in the bound on queue size given in Theorem 2, the B_2 term is proportional to T, i.e., the delay increases with T. However, reasonable delay values are possible with appropriate choices of T, e.g., if we choose T to be 240 time slots (1 hour), SAVE gives an average delay of 14.8 time slots (3.7 minutes). From this set of results we can see that for delay tolerant workloads, SAVE would take infrequent actions on server activation/sleep (once in an hour

[2] According to [18], a large fraction (about 80%) of data center operators do not identify idle servers on a daily basis.

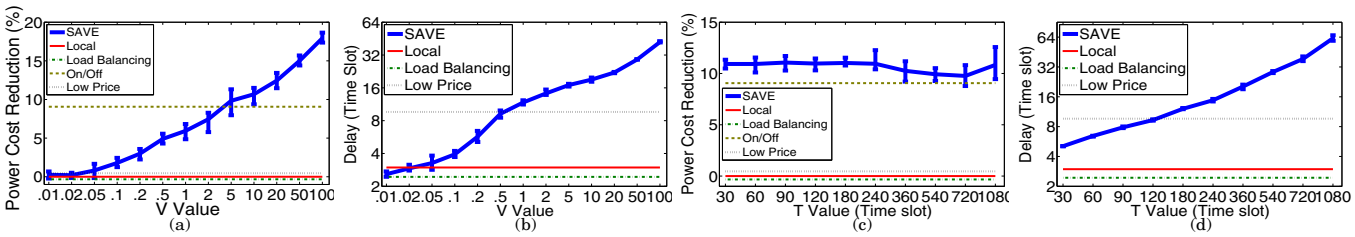

Fig. 3. Average power cost and delay of all schemes under different V and T values

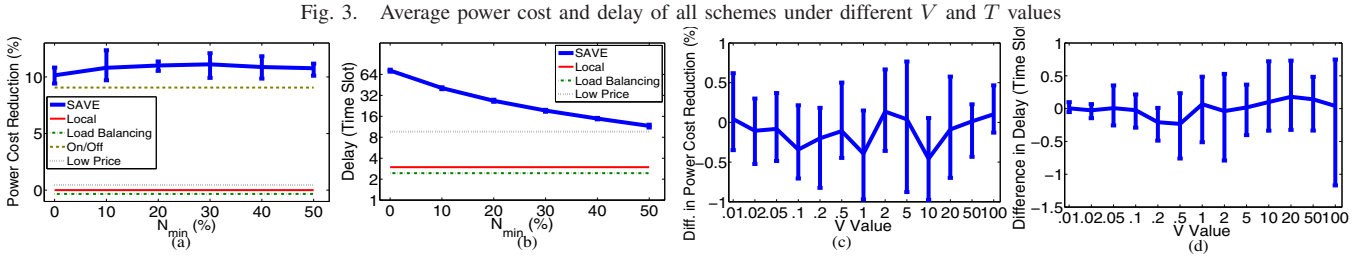

Fig. 4. Average power cost and delay of all schemes under different N_{min} values and robustness test results

or less) and still achieve significant power cost reduction.

B. The impact of N_{min}

In this set of experiments we keep V and T values unchanged, but vary N_{min} values from 0 to 50% of the number of servers in a data center. The results are depicted in Figures 4(a) and (b). Figure 4(b) indicates that increasing N_{min} improves delay performance, e.g., when it increases from 0 to 20% of the number of servers, the average delay decreases significantly, from about 72.5 to 25.9 time slots. At the same time, as shown in Figure 4(a), the effect of N_{min} on power cost reduction is relatively small. This makes intuitive sense. When N_{min} grows larger, more servers are activated regardless of job arrivals, providing more slots to serve jobs, thus reducing average delay. On the other hand, adding more active servers reduces the service rate of each server, which compensates for the extra power consumed by added servers.

C. Robustness Characteristics

As mentioned in Section II-E, our SAVE algorithm needs to know the amount of workload of each job. In practice, when this is not available, we use estimated values. In this set of experiments we explore the influence of estimation errors on the performance of SAVE. To do this, for each arriving job, we add a random estimation error ($\pm 50\%$, uniformly distributed) to the amount of workload it contains. This gives us one error data set for each arrival data set. We run SAVE on these data sets, but let SAVE make all decisions on control variables based on the amount of workload with estimation errors. Only when a job get served does the server know the exact amount of workload it actually contains.

We run experiments for all 5 pairs of data sets for different V values, and compare the results to the results we obtained using the original arrival data sets. In Figures 4(c) and (d), we use the results on data sets **without** estimation errors as the baseline, and show the differences in power cost reduction percentage and delay (in time slots) due to injected estimation errors. From Figure 4(c) we can see that for all V values we experimented with, the difference(due to errors) in power cost reduction is between -1.0% and 0.7%. As shown in Figure 4(d), estimation errors result in changes in average delay, but only in the range of -1.2 to 0.9 time slots.

To conclude, SAVE is robust to workload estimation errors.

D. Actual Power Consumption of SAVE

SAVE is designed to reduce the *cost* of power in geographically distributed data centers, as this is one major concern for large computational facility providers. At the same time, with more attention paid to the social and ecological impact of large computational infrastructures, it is also desirable to consider environmental friendly approaches, i.e., while reducing the cost of power, it is also desirable to reduce the *actual consumption* of power. To this end, we record the actual power consumption of all simulated schemes. In Figure 5 we show the percentage of average power consumption reduction by SAVE with different V values, relative to the Local Computation scheme. Figure 5 illustrates that with V values ramping from

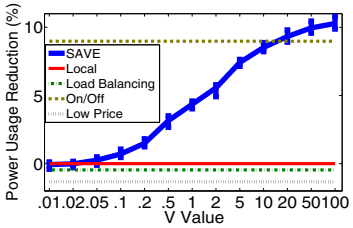

Fig. 5. Differences in average power usage reduction for different V values

0.01 to 100, the actual power consumption reduction goes from about 0.1% to 10.3%. When $V = 10$, the reduction is around 8.7%. This indicates that SAVE is environmentally friendly, in a sense that, while it reduces power cost, it also reduces actual power consumption significantly. As a comparison, the Low Price scheme is not environmentally friendly – although it reduces power cost (see Figure 3(a)), it consumes more power than the Local Computation scheme.

VII. Related Work

As mentioned in Section I, work on power cost reduction can be classified into three broad categories – single server, single data center, and multiple data centers. For a single server, researchers have proposed scheduling algorithms to

minimize power consumption subject to job deadlines [20], or minimize average response time subject to a power constrain [21]. Wierman et al. use dynamic CPU speed scaling to minimize weighted sum of mean response time and power consumption [5]. A survey of work on single server power cost reduction is given in [22]. For a data center, Gandhi et al. provide management policies that minimize mean response time under a total power cap [9] or minimize the product to response time and power cost [7]. Chen et al. propose solutions based on queueing models and control theory to minimize the server energy as well as data center operational costs [8]. Lin et al. design an online algorithm to minimize a convex function of power usage and delay that is 3-competitive [4]. SAVE differs from these work in three ways: (i) it leverages spatio-temporal differences in job arrivals and power prices at different geographic locations to achieve power cost reduction; (ii) all work mentioned above except [21] and [20] assume closed form convex expressions for service delay or convex delay-cost functions, whereas SAVE does not make these assumptions as they may not always hold, especially for delay tolerant jobs; (iii) SAVE does not rely on predictions of workload arrival processes, as [8] and [4] do.

Power cost reduction across multiple data centers is an area of active research. Qureshi et al. proposed the idea of exploiting spatial diversity in power prices to reduce cost by dynamically routing jobs to data centers with lower prices [2]. They also provide a centralized heuristic for doing so that is similar to the Low Price scheme we evaluated in Section VI. Rao et al. provide an approximation algorithm for minimizing the total power cost subject to an average delay constraint [19], while Liu et al. designed load balancing algorithms to minimize a metric that combines power and delay costs [3]. Both papers make routing and server on/off decisions based on predictions on arrival rates and closed form convex expressions for average delay. [6] makes control decisions at three levels – server, data center and across multiple data centers – in one time slot by solving a deterministic convex optimization problem. All these work have a work conserving scheduler and only exploit the spatial differences in job arrivals and power prices. Also, they work on a single time scale. In contrast, SAVE exploits both the spatial and temporal differences in job arrivals and power prices by using a non work conserving scheduler. This leads to greater power cost reductions when serving delay-tolerant workloads. Moreover, it works on two time scales to reduce the server on/off frequency.

The Lyapunov optimization technique that we use to design SAVE was first proposed in [23] for network stability problems. It was generalized in [11] for network utility optimization problems. Recently, Urgaonkar et al. used this technique to design an algorithm for joint job admission control, routing, and resource allocation in a virtualized data center [24]. However, they consider power reduction in a single data center only. To the best of our knowledge, our work is the first to apply a novel two time scale network control methodology to distributed workload management for geographically distributed data centers.

VIII. Conclusions

In this paper, we propose a general framework for power cost reduction in geographically distributed data centers. Our approach incorporates routing and server management actions on individual servers, within a data center, and across multiple data centers, and works at multiple time scales. We show that our approach has provable performance bounds and is especially effective in reducing power cost when handling delay tolerant workloads. We also show that our approach is robust to workload estimation errors and can result in significant power consumption reductions.

References

[1] www.gizmodo.com/5517041/.

[2] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," in *SIGCOMM*, 2009.

[3] Z. Liu, A. Wierman, S. Low, and L. Andrew, "Greening geographical load balancing," in *ACM SIGMETRICS*, 2011.

[4] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *IEEE INFOCOM*, 2011.

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

[6] R. Stanojevic and R. Shorten, "Distributed dynamic speed scaling," in *IEEE INFOCOM*, 2010.

[7] A. Gandhi, V. Gupta, M. Harchol-Balter, and A. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Perform. Eval.*, vol. 67, pp. 1155–1171, November 2010.

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

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

[10] http://googleenterprise.blogspot.com/2010/03/disaster-recovery-by-google.html.

[11] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Foundations and Trends in Networking*, vol. vol. 1, no. 1, pp. 1-149, 2006.

[12] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Power cost reduction for delay tolerant workloads in distributed data centers: a two time scale approach," University of Southern California Computer Sciences Department, Tech. Rep., July 2011.

[13] L. Huang and M. J. Neely, "Max-weight achieves the exact $[o(1/v), o(v)]$ utility-delay tradeoff under markov dynamics," *arXiv:1008.0200v1*, 2010.

[14] Federal Energy Regulatory Commission www.ferc.gov.

[15] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *EuroSys*, 2010.

[16] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *OSDI*, 2010.

[17] www.google.com/corporate/green/.

[18] "Unused servers survey results analysis," http://www.thegreengrid.org/.

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

[20] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *IEEE FOCS*, 1995.

[21] K. Pruhs, P. Uthaisombut, and G. Woeginger, "Getting the best response for your erg," *ACM Trans. Algorithms*, vol. 4, pp. 1–17, July 2008.

[22] S. Albers, "Energy-efficient algorithms," *Commun. ACM*, vol. 53, pp. 86–96, May 2010.

[23] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. on Automatic Control, vol. 37, no. 12, pp. 1936-1949*, Dec. 1992.

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