Power Cost Reduction in Distributed Data Centers: A Two-Time-Scale Approach for Delay Tolerant Workloads

Yuan Yao, Longbo Huang, Abhishek B. Sharma, Leana Golubchik, and Michael J. Neely

Abstract—This paper considers a stochastic optimization approach for job scheduling and server management in large-scale, geographically distributed data centers. Randomly arriving jobs are routed to a choice of servers. The number of active servers depends on server activation decisions that are updated at a slow time scale, and the service rates of the servers are controlled by power scaling decisions that are made at a faster time scale. We develop a two-time-scale decision strategy that offers provable power cost and delay guarantees. The performance and robustness of the approach is illustrated through simulations.

Index Terms—Power management, data center, stochastic optimization, performance analysis

1 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 ($\approx$1 million), Microsoft ($\approx$200K), Akamai (60-70K), INTEL($\approx$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 a considerable monetary saving.

Extensive research has been carried out to reduce power cost in data centers, for example, [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: First, 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. Second, 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. Third, at the interdata 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.

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* trade-off 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 Fig. 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.

Our goal then is to minimize power cost by deciding: 1) how to distribute the workload from the front-end servers to the back-end clusters; 2) how many servers to activate at each back-end cluster at any given time; and 3) how to set the service rates (or CPU power levels) of the back-end servers.

There are two challenging aspects of this problem: 1) different control actions act as different time scales—power scaling can be performed much faster than server activation; and 2) both power prices and workload are stochastic in nature and can be hard to predict accurately. To address these challenges, we design an algorithm that makes decisions over two time scales. The algorithm uses Lyapunov optimization concepts from [8] and offers explicit performance guarantees in these stochastic settings.

Our proposed solution exploits temporal and spatial variations in the workload arrival process (at the front-end

- *Y. Yao is with the University of Southern California, Los Angeles, SAL200, 941 Bloom Walk, Los Angeles, CA 90089.*
- *L. Golubchik is with the University of Southern California, Los Angeles, SAL 226, 941 Bloom Walk, Los Angeles, CA 90089.*
- *M.J. Neely is with the University of Southern California, Los Angeles, 3740 McClintock Ave., Room 520, Los Angeles, CA 90089.*
- *L. Huang is with Tsinghua University, Room 1-208, FIT Building, Beijing 100084, P.R. China.*
- *A. Sharma is with the NEC Laboratories America, Inc., 4 Independence Way, Suite 200, Princeton, NJ 08540.*

Manuscript received 21 Apr. 2012; revised 5 Dec. 2012; accepted 13 Dec. 2012; published online 21 Dec. 2012.
Recommended for acceptance by Y.C. Hu.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2012-04-0400.
Digital Object Identifier no. 10.1109/TPDS.2012.341.

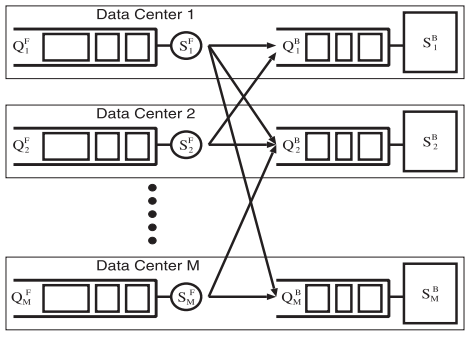


Fig. 1. A model of M geographically distributed data centers.

servers) and the power prices (at the back-end clusters) to reduce power cost. It also facilitates a *cost versus 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 cloud computing and data intensive MapReduce jobs. Today, MapReduce-based applications are used to build a wide array of web services, for example, search, data analytics, social networking, etc. For example, it is indicated in [9] that Google often has a large number of "long duration" jobs running on back-end servers. These jobs take from several minutes to more than 20 hours and thus are relatively delay tolerant.

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 versus delay trade-off in geographically distributed data centers (see Sections 2 and 3).
- 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 (see Section 4).
- Through simulations based on real-world power price data sets, we show that our approach can reduce the power cost by as much as 18 percent, even for state-of-the-art server power consumption profiles and data center designs (see Section 6). We also show that our approach is environment friendly, in the sense that it not only reduces power cost but also the actual power usage (see Section 6.5).
- We demonstrate the robustness of our approach to errors in estimating data center workload both analytically as well as through simulations (see Section 6.2).
- We evaluate the performance of our approach with data sets of different workload intensities and show that our approach performs better when workload intensity is lower. It is able to reduce power cost by close to 60 percent when average system load is around 26 percent. (see Section 6.3).
- Further evaluations show that our approach is able to reduce power cost for non-i.i.d. arrival data sets with various autocorrelation levels (see Section 6.4).

2 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 $\mathcal{D} = \{D_1, \ldots, D_M\}$, where the system operates in slotted time, i.e., $t = 0, 1, 2, \ldots$. 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.

2.1 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 $\boldsymbol{A}(t) = (A_1(t), \ldots, A_M(t))$ denotes the arrival vector. In our analysis, we first assume that $\boldsymbol{A}(t)$ are i.i.d. every time slot with $\mathbb{E}\{\boldsymbol{A}(t)\} = \boldsymbol{\lambda} \triangleq (\lambda_1, \ldots, \lambda_M)$. 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.

2.2 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 $\boldsymbol{\mu}_i(t) = (\mu_{i1}(t), \ldots, \mu_{iM}(t))$ 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, for example, 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 $k = 0, 1, \ldots$, 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 $N_i(t) \in \{N_{min}^i, N_{min}^i + 1, N_{min}^i + 2, \ldots, N_i\}$, 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: 1) activating servers typically costs a nonnegligible amount of time and power, and 2) frequently switching back and forth between active and sleep states can result in reliability problems [10]. 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

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

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 $b_i(t), 0 \le b_i(t) \le b_{max}$, 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 2.5.

2.3 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 $P_i(N_i(t_T), b_i(t)) \le P_{max}$ for all t and i. Such power consumption will in turn incur some monetary cost for the data centers, of the form "power $\times$ 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 $\boldsymbol{p}(t) = (p_1(t), \ldots, p_M(t))$ varies every $T_1 \ge 1$ 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 $\mathcal{P}_i = \{p_i^1, \ldots, p_i^{|\mathcal{P}_i|}\}$. 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 Fig. 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 $\sum_i f_i(t) \le f_{max}$ for all t.

2.4 The Data Center Power Cost Minimization (PCM) Problem

Let $\boldsymbol{Q}(t) = (Q_i^F(t), Q_i^B(t), i = 1, \ldots, M)$, $t = 0, 1, \ldots$, 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 queuing dynamics:

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

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

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 4. Throughout the paper, we use the following definition of queue stability:

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

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

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

Here, $f_i^{\Pi}(\tau)$ denotes the power cost incurred by D_i under 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 PCM problem in the remainder of the paper.

2.5 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], [11]). If the N activated servers run at service rates $b_1, b_2, \ldots, b_N$, 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

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

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 [12]. 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 4, we show that even if the estimation is not accurate, we can still prove bounds on the performance of our approach. Moreover, in Section 6.2, 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 W when in sleep state, as compared to 150 W in idle state. This indicates that our assumption is reasonable. We also assume that servers can be activated and deactivated immediately. This assumption involves two issues: 1) regarding server wake up: We note that waking up from sleep takes around 60 seconds. During these 60 seconds, the server cannot perform any work. 2) Regarding server deactivation: We assume that running jobs on these servers need to be migrated to active servers, but this takes some time and introduces additional power cost. Thus, these operations 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 delay and cost incurred by such actions is amortized over the relatively long period during which the server is active. The effect of T is further discussed in Section 6, where we give experimental results.

3 The StochAstic Power redUction schEme (SAVE) Algorithm

We solve PCM through our 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.

3.1 SAVE's Control Algorithms

SAVE's three control actions are the following:

- *Front-end routing.* In every time $t = kT, k = 0, 1, \ldots$, each D_i solves for $\mu_{ij}(t)$ to maximize

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

  where $\boldsymbol{\mu}_i(t) = (\mu_{i1}(t), \ldots, \mu_{iM}(t)) \in \mathcal{R}_i$. Then, in every time slot $\tau \in [t, t+T-1]$, D_i sets $\mu_{ij}(\tau) = \mu_{ij}(t)$ and route up to $\mu_{ij}(\tau)$ amount of workload to the back-end cluster at D_j, $1 \le j \le M$.
- *Back-end server management.* At time slot $t = kT$, D_j chooses $N_j(t) \in [N_{min}^j, N_j]$ to minimize

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

  Then, D_j uses $N_j(t)$ servers over time $[t, t+T-1]$. At every time slot $\tau \in [t, t+T-1]$, each D_j chooses $b_j(\tau)$ (note that $N_j(t)$ is determined at time slot t) to minimize

$$V f_j(\tau) - Q_j^B(t) N_j(t) b_j(\tau). \tag{7}$$

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

Note that V is a parameter of SAVE that controls the tradeoff between power cost and delay performance, as detailed in Section 4.2. SAVE works at two different time scales. The front-end routing and number of active servers are determined every T slots, while back-end servers' service rates are updated each slot. This two-time-scale mechanism is important from a practical perspective because activating servers usually takes much longer than power scaling.

3.2 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 6.1.

Second, SAVE can provide opportunities for bandwidth cost reductions because 1) it provides an explicit upper bound on the workload sent from S_i^F to $\mathcal{S}_j^B$, and 2) these routing decisions remain unchanged for T time slots. If T is large, this can provide opportunities for data centers to optimize data transmission ahead of time to reduce bandwidth cost. As suggested in [2], content distribution networks like Akamai can incur significant bandwidth costs. Incorporating bandwidth costs into our model is part of future work.

3.3 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 computes 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. It is obtained as follows: At every $t = kT$, $k \ge 0$, we use the empirical distribution of prices over a time window of size L to compute the expectation. Specifically, if $t \ge L$, 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

$$\hat{\pi}_i(p) \triangleq \frac{n_p^i(t, L)}{L}$$

as the probability $\pi_i(p)$ for estimating the expectations. Since all $p_i(t)$ only take finitely many values, we have

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

Therefore, as we increase the number of samples, the estimation becomes better and better.

Now we can solve for $N_j(t)$ through an optimization of (6), i.e., for each candidate choice of the number of active servers $N_j(t) \in [N_{min}^j, N_j]$, we define

$$g(\hat{N}_j) \triangleq \sum_p \left\{ \hat{\pi}_j(p) \sum_{\tau=t}^{t+T-1} \min_{b_j(\tau)\in[0,b_{max}]} \left\{ Vp \cdot P\big(\hat{N}_j, b_j(\tau)\big) - Q_j^B(t)\hat{N}_j b_j(\tau) \right\} \right\}. \tag{9}$$

Note that in (9), each minimization has only a single variable $b_j(\tau)$, which is simple to carry out. The next step is to choose $N_j(t)$ from $[N_{min}^j, N_j]$ that minimizes $g(\hat{N}_j)$.

4 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 [13]. 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 it to two 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 performance.

4.1 Algorithm Design

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

$$L(t) \triangleq \frac{1}{2}\sum_{i=1}^{M}\left(\big[Q_i^F(t)\big]^2 + \big[Q_i^B(t)\big]^2\right). \tag{10}$$

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

$$\Delta_T(t) \triangleq \mathbb{E}\{L(t+T) - L(t)|\boldsymbol{Q}(t)\}. \tag{11}$$

Following the Lyapunov optimization approach, we add the expected power cost over T slots (i.e., a penalty function), $\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_j f_j(\tau)\}$, to (11) 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.

Lemma 1. *Let $V > 1$ 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*

$$\begin{aligned}
&\Delta_T(t) + V\mathbb{E}\left\{\sum_{\tau=t}^{t+T-1}\sum_j f_j(\tau)|\boldsymbol{Q}(t)\right\} \\
&\le B_1 T + V\mathbb{E}\left\{\sum_{\tau=t}^{t+T-1}\sum_j f_j(\tau)|\boldsymbol{Q}(t)\right\} \\
&\quad - \mathbb{E}\left\{\sum_{\tau=t}^{t+T-1}\sum_i Q_i^F(\tau)\left[\sum_j \mu_{ij}(\tau) - A_i(\tau)\right]|\boldsymbol{Q}(t)\right\} \\
&\quad - \mathbb{E}\left\{\sum_{\tau=t}^{t+T-1}\sum_j Q_j^B(\tau)\big[N_j(t)b_j(\tau) - \sum_i \mu_{ij}(\tau)\big]|\boldsymbol{Q}(t)\right\}.
\end{aligned} \tag{12}$$

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

Proof. See supplementary document, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.341. □

The main design principle in Lyapunov optimization is to choose control actions that minimize the R.H.S. of (12). However, for any slot t, this requires *prior* knowledge of the future queue backlogs ($\boldsymbol{Q}(t)$) over the time frame $[t, t+T-1]$. $\boldsymbol{Q}(\tau)$ depends on the job arrival processes $A_i(\tau)$ and SAVE's decisions $\mu_{ij}(t), b_i(t)$, and $N_i(t)$ (that depend on time varying power prices). Hence, minimizing the R.H.S. of (12) 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 $t < \tau \le t+T-1$. However, the simplification forces a "loosening" of the upper bound on the drift-plus-penalty term as shown in the following lemma.

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

$$\begin{aligned}
&\Delta_T(t) + V\mathbb{E}\left\{\sum_{\tau=t}^{t+T-1}\sum_j f_j(\tau)|\boldsymbol{Q}(t)\right\} \\
&\le B_2 T + \mathbb{E}\left\{\sum_{\tau=t}^{t+T-1}\sum_j Q_i(t)A_i(\tau)|\boldsymbol{Q}(t)\right\} \\
&\quad - \mathbb{E}\left\{\sum_{\tau=t}^{t+T-1}\sum_i\sum_j \mu_{ij}(\tau)\big[Q_i^F(t) - Q_j^B(t)\big]|\boldsymbol{Q}(t)\right\} \\
&\quad + \mathbb{E}\left\{\sum_{\tau=t}^{t+T-1}\sum_j \big[Vf_j(\tau) - Q_j^B(t)N_j(t)b_j(\tau)|\boldsymbol{Q}(t)\right\}.
\end{aligned} \tag{13}$$

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

Proof. See the online supplementary document. □

Comparing (13) 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 (13).

4.2 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 $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_M)$, 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 $N_i(t), i = 1, \ldots, M$ 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 $k = 0, 1, 2, \ldots$:*

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

Proof. It can be proven using Caratheodory's theorem in [13]. 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 $\boldsymbol{\lambda}+\epsilon\mathbf{1}\in\boldsymbol{\Lambda}$; then, under the SAVE algorithm, we have*

$$\overline{Q}_T\triangleq\limsup_{t\to\infty}\frac{1}{K}\sum_{k=0}^{K-1}\sum_{i=1}^{M}\mathbb{E}\{Q_i^F(kT)+Q_i^B(kT)\}\le\frac{B_2+Vf_{max}}{\epsilon},\tag{14}$$

$$f_{av}^{\mathrm{SAVE}}\triangleq\limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\sum_{i=1}^{M}\mathbb{E}\{f(\tau)\}\le f_{av}^{*}+\frac{B_2}{V}.\tag{15}$$

Here, f_{av}^{*} is the optimal cost defined in Section 2 and $\mathbf{1}$ denotes the vector of all 1s.

Proof. See the online supplementary document. □

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 $\boldsymbol{\lambda}+\epsilon\mathbf{1}\in\boldsymbol{\Lambda}$. 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 $|\hat{Q}_i^F(t)-Q_i^F(t)|\le c_e$ and $|\hat{Q}_i^B(t)-Q_i^B(t)|\le c_e$. Then, under the SAVE algorithm, we have*

$$\overline{Q}_T\triangleq\limsup_{t\to\infty}\frac{1}{K}\sum_{k=0}^{K-1}\sum_{i=1}^{M}\mathbb{E}\{Q_i^F(kT)+Q_i^B(kT)\}\le\frac{B_3+Vf_{max}}{\epsilon},\tag{16}$$

$$f_{av}^{\mathrm{SAVE}}\triangleq\limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\sum_{i=1}^{M}\mathbb{E}\{f(\tau)\}\le f_{av}^{*}+\frac{B_3}{V}.\tag{17}$$

Here, f_{av}^{*} is the optimal cost defined in Section 2, and $B_3=B_2T+2MTc_e(\mu_{max}+A_{max}+N_{max}b_{max}+M\mu_{max})$.

Proof. See the online supplementary document. □

By comparing inequalities (17) and (15), 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 (16) and (18)). Hence, SAVE works even with inaccurate queue backlog information, but its robustness is achieved at the expense of a power cost versus delay trade-off. We also note that according to (16) and (17), given the same V and c_e values, smaller ϵ values lead to larger average queue sizes and higher costs. ϵ is a measure of the system's "slackness." Therefore, the same level of inaccuracy will cause greater performance degradation of SAVE when the system is heavily loaded. We further demonstrate SAVE's robustness using simulations in Section 6.

4.3 Extension to Non-i.i.d. Arrival and Power Prices

Theorems 2 and 3 are obtained with the assumption that $\boldsymbol{A}(t)$ is i.i.d. We note that they can indeed be extended to the case when $\boldsymbol{A}(t)$ and $\boldsymbol{p}(t)$ are Markovian. More specifically, regarding the performance of SAVE, we can prove the following theorem.

Theorem 4. *When $\boldsymbol{A}(t)$ is Markovian, then under the SAVE algorithm, we have*

$$\overline{Q}_T\triangleq\limsup_{t\to\infty}\frac{1}{K}\sum_{k=0}^{K-1}\sum_{i=1}^{M}\mathbb{E}\{Q_i^F(kT)+Q_i^B(kT)\}\le O(V),\tag{18}$$

$$f_{av}^{\mathrm{SAVE}}\triangleq\limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\sum_{i=1}^{M}\mathbb{E}\{f(\tau)\}\le f_{av}^{*}+O\Big(\frac{1}{V}\Big).\tag{19}$$

Here, f_{av}^{*} is the optimal cost defined in Section 2.

The theorem can be proven using a similar argument as in [14]. The sketch of the proof is provided as follows: We first redefine $\boldsymbol{A}_T(k)=(\boldsymbol{A}(kT),\ldots,\boldsymbol{A}((k+1)T-1))$ and $\boldsymbol{p}_T(k)=(\boldsymbol{p}(kT),\ldots,\boldsymbol{p}((k+1)T-1))$ as the frame arrival state and frame price state. Since both $\boldsymbol{A}(t)$ and $\boldsymbol{p}(t)$ evolve according to a finite-state Markov chain, it can be seen that $\boldsymbol{z}(t)\triangleq(\boldsymbol{A}_T(k),\boldsymbol{p}_T(k))$ evolves according to a finite-state Markov chain and has a steady-state distribution. Now choose any state $\boldsymbol{z}_T^{*}$. We first show that (12) holds without the expectation over a single frame. Then, assuming $\boldsymbol{z}_T(k)=\boldsymbol{z}_T^{*}$, we define a W-frame Lyapunov drift,

$$\Delta_T^W(t)=\mathbb{E}\{L((k+W)T)-L(kT)\mid\boldsymbol{Q}(kT)\},\tag{20}$$

where W is a random time such that $(k+W)T$ is the next time $\boldsymbol{z}_T(k)=\boldsymbol{z}_T^{*}$. After that, we show that the right-hand side of $\mathbb{E}\{\Delta_T^W(t)\}$ can be bounded by the minimum power cost $\mathbb{E}\{W\}Tf_{av}^{*}$, by using a renewal theory argument and duality. After this, the theorem can be proven as in the proof of Theorem 2. Similarly, we can also prove a theorem that resembles Theorem 3 with $\boldsymbol{A}(t)$ and $\boldsymbol{p}(t)$ being Markovian.

5 Experimental Setup

The goal of this experimental study is to evaluate the performance of SAVE. Our evaluation scenario consists of seven data centers at different geographic locations. Next, we describe the main components of our simulations: the electricity prices and workloads at different data centers,

TABLE 1
Distribution of Job Sizes

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

the system parameters, and alternate schemes against which we compare SAVE.

5.1 Data Sets

Electricity prices. We downloaded the hourly electricity prices for seven hubs at different geographic locations from [15]. 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, for example, 5-minute intervals [2]. However, since we had access to only the hourly prices, we use interpolation to generate prices at 5-minute intervals.

Workload. We generate synthetic workload that mimics MapReduce jobs, according to the published statistics on MapReduce usage on Facebook [16]. 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: 1) all tasks belonging to a job have the same processing time; tasks from different jobs can have different processing times; 2) jobs can be served in any of the seven data centers; and 3) all the tasks from the same job must be served at the same back-end cluster. Regarding assumption 1, 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 [17]. Hence, assumption 1 is not a significant oversimplification. As explained in Section 2.5, we believe assumption 2 is also reasonable. Assumption 3 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 0 and 60 seconds with the "job size" distribution (i.e., number of tasks per job) given in Table 1; these distributions (job execution time and "job size") correspond to data reported in [16].

We generated five groups of delay tolerant workloads. Each group consists of seven 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 [16].

In addition to that, we generate 10 groups of workloads with Poisson arrival processes: 5 with arrival rate 5 jobs per time slot, 5 with 10 jobs per time slot. These workloads, combined with those mentioned above, are used to study performance of SAVE under different workload intensities. Finally, we generate three groups of non-i.i.d. workloads with the same job arrival rate but different autocorrelation. The average autocorrelation (lag 1) of these three groups are 0.98, 0.50, and 0.29, respectively. They are used to evaluate the impact of traffic correlations on SAVE.

5.2 Experimental Settings

Power cost function. SAVE can handle a wide range of power cost functions including nonconvex functions. In our experiments, we model power consumption $P(N_i, b_i)$ as

$$P(N_i(t), b_i(t)) = \left(N_i(t) \left(\frac{b_i(t)^\alpha}{A} + P_{idle} \right) \right) \cdot PUE \qquad (21)$$

In (21), 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 $\alpha = 3$, $P_{idle} = 150$ W, and A such that the peak power consumed by a server is 250 W. The model (21) and all its parameters are based on the measurements reported in [11]. The PUE term accounts for additional power usage (such as cooling). According to [18], 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 achieves larger power cost reductions when PUE is higher.

System parameters. We set N_i, the number of servers in data center i, to be 1,000 for all seven data centers. Each server can serve up to 10 tasks at the same time. Since average running time of a task is 30 seconds, $b_i(t)$ can be chosen between 0 and 5 tasks per time slot. With a 15 jobs per slot arrival rate, this gives an average server utility of about 78 percent. We set the bandwidth between the front-end servers and the back-end clusters to a large value, based on the real-world practice [12]. Hence, a large amount of workload can be sent from one data center to another within one time slot. We vary N^i_{min} across a range of values to explore its impact on SAVE's performance.

5.3 Schemes Simulated for Comparison

SAVE is compared to the following work-conserving schemes, representing the current data center practices or heuristics proposed by others.

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

Load balancing. The amount of workload routed from D_i to D_j, μ_{ij}, is proportional to the service capacity of D_j ($b_{max} \cdot N_i$), 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

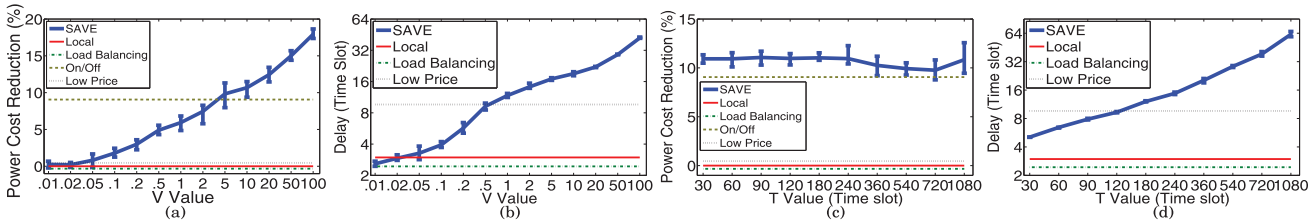


Fig. 3. Average power cost and delay of all schemes under different V and T values.

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 up at all times.[1] However, we assume that the service rates of the servers can be tuned every slot to optimize power cost. And we simulate the best they can do in reducing power cost. We also simulate the following scheme that deactivates idle servers.

Instant on/off. Here, the routing decisions between front-end servers and back-end clusters are exactly the same as in scheme Load Balancing. 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. In particular, we assume that data center D_i is able to chooses optimal $b_i(t)$ and $N_i(t)$ according to $Q_i^B(t)$in each and every time slot t. 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).

6 Experimental Results

6.1 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 other parameters fixed). For power cost reduction results, we use scheme Local Computation as a baseline. For all other schemes, we show the percentage of average power cost reduction as compared to scheme Local Computation. Specifically, let PC_X denote the average power cost of scheme X. $\frac{PC_{L.C.}-PC_X}{PC_{L.C.}} \times 100$ is used to quantify power cost reduction of scheme X. ($L.C.$ is short for Local Computation.) For delay results, we show the schemes' average delay (in number of time slots). Delay results of scheme On/Off are omitted as they are nearly identical to those of scheme Load Balancing—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 (1 hour) and run experiments with different V values. The results are shown in Figs. 3a and 3b. From Fig. 3a, we can see that as V goes from 0.01 to 100, the power cost reduction grows from an average of around 0.1 percent to about 18 percent. Scheme On/Off achieves power cost reduction of about 9.7 percent. If we choose $V \geq 5$, then SAVE results in larger power cost reductions than scheme On/Off (which is impractical). This is because 1) our approach considers differences in power prices across different data centers, and 2) our approach is not work conserving and can adjust service rates at different data centers according to power prices. We also note that scheme Low Price gives a small power cost reduction ($\approx 0.5\%$)—i.e., sending more workload to data centers with low power prices in a greedy fashion is not very effective in reducing power cost. In Fig. 3b, 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 leads to larger delay as well as larger power cost reductions. In general, V in SAVE controls the trade-off between delay and power cost.

We fix V to be 10 and vary T from 30 time slots (7.5 minutes) to 1,080 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 [20]. Also, when setting $T \leq 30$, servers may be powered up and down very frequently. This is not efficient, as discussed in Section 2.5.) Corresponding results of the different schemes are shown in Figs. 3c and 3d. From Fig. 3c, 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 percent when T varies from 30 to 1,080 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 Fig. 3d. In the extreme case, when $T = 1{,}080$ 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, for example, if we choose $T = 240$ 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

1. Idle servers are not indentified by about 80 percent of data center operators [19].

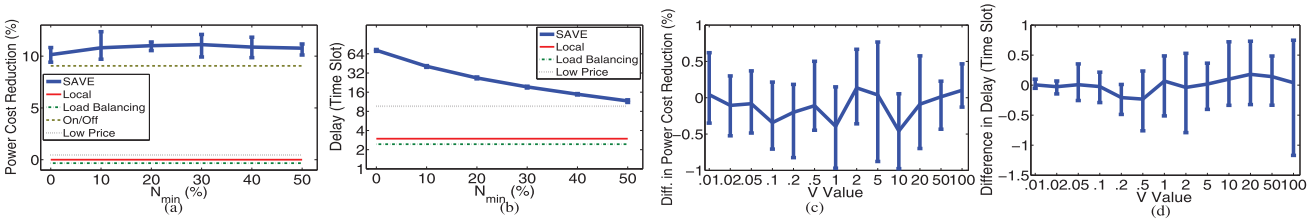


Fig. 4. Average power cost and delay of all schemes under different N_{min} values and robustness test results.

infrequent actions on server activation/sleep (once in an hour or less) and still achieve significant power cost reduction.

In the next set of experiments, we consider the impact of N_{min}. We keep V and T values fixed, but vary N_{min} values from 0 to 50 percent of N_i. The results are depicted in Figs. 4a and 4b. Fig. 4b indicates that increasing N_{min} improves delay performance, for example, when N_{min} is increased to 20 percent of N_i, the average delay decreases from $\approx$72.5 to $\approx$25.9 time slots. At the same time, as shown in Fig. 4a, the effect of N_{min} on power cost reduction is relatively small. This makes intuitive sense. When N_{min} increases, more servers are up 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.

6.2 Robustness Characteristics

As mentioned in Section 2.5, SAVE 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 job, we add a random estimation error ($\pm 50\%$, uniformly distributed) to the amount of workload it contains. We run SAVE on these data sets, but let SAVE make all control decisions based on the workload with estimation errors. Only when a job get served does the server know the actual amount of workload it contains.

We run experiments for all five groups of data sets for different V values and compare the results to those obtained using the original arrival data sets. In Figs. 4c and 4d, we use the results on data sets *without* estimation errors as the baseline and show the differences in power cost reduction and delay due to injected estimation errors. Fig. 4c indicates that for all V values we experimented with, the difference (due to errors) in power cost reduction is between -1.0 and 0.7 percent. As shown in Fig. 4d, changes in average delay caused by estimation errors are small (between -1.2 and 0.9 time slots).

To conclude, SAVE is robust to workload estimation errors.

6.3 The Impact of Workload Intensity

Here, we consider workloads with different workload intensity. In this set of experiments, we keep T and N_{min} values unchanged and run SAVE with different V values on data sets with different arrival rates. With low, medium, and high arrival rates, the average system utilization is around 26, 52, and 78 percent, respectively. We group experimental results on data sets with the same arrival rate together. The results are depicted in Figs. 5a and 5b. For comparison, we also plot the average power cost reduction and delay of scheme On/Off.

From Figs. 5a and 5b, we can see that as workload intensity increases, SAVE results in less power cost reduction and higher delay with the same V value. In particular, for data sets with low workload intensity, SAVE can reduce power cost up to almost 60 percent. But with high workload intensity, the same V value only brings in 18 percent reduction in power cost. This makes sense. With small workload intensity, more servers can be deactivated to save power. On the other hand, with higher workload intensity, delay increases, and there is less "room" for SAVE to delay jobs for better performance. We can also observe that if we set V to be larger than 5, SAVE outperforms scheme On/Off in power cost reduction in most data sets.

6.4 Non-i.i.d. Workload

We use three groups of workloads with different levels of autocorrelation to evaluate the performance of SAVE on non-i.i.d. arrivals. Power cost reduction of SAVE for these workloads is depicted in Fig. 5c. The curve indicates that SAVE results in higher reduction as V increases. The bars show that SAVE offers similar performance for all data sets. In fact, with the same V value, the difference in relative power cost reduction is at most 1 percent.

In Fig. 5d, average delay of SAVE are depicted for each of these workloads. From the figure, we can observed that when V is small, higher autocorrelation workloads results in higher delay. This is because jobs arrives more in batches

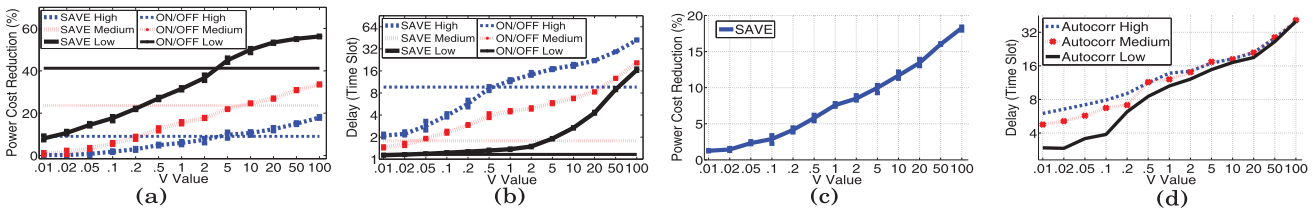


Fig. 5. Average power cost and delay of SAVE with different workload intensity and autocorrelations.

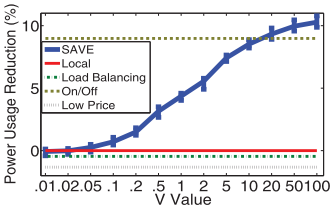


Fig. 6. Differences in average power usage reduction for different V values.

when autocorrelation is high, and more in spread out manner when autocorrelation is low. With increased V, SAVE has larger flexibility of delaying jobs for better performance. Thus, delay performance become similar for all three data sets.

In conclusion, SAVE is able to provide similar level of power cost reduction regardless of autocorrelations in arrival. But its performance in delay may vary when V is small.

6.5 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 *consumption* of power. To this end, we record the power consumption of all simulated schemes. In Fig. 6, we show the percentage of average power consumption reduction by SAVE with different V values, relative to scheme Local Computation. Fig. 6 illustrates that with V values ramping from 0.01 to 100, the power consumption reduction goes from about 0.1 to 10.3 percent. When $V = 10$, the reduction is around 8.7 percent. This indicates that SAVE is environmentally friendly—while it reduces power cost, it also reduces power consumption significantly. As a comparison, the Low Price scheme is not environment friendly. Although it reduces power cost (see Fig. 3a), it consumes more power than scheme Local Computation.

7 Related Work

As mentioned in Section 1, 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 algorithms to minimize power consumption subject to job deadlines [21], minimize average response time subject to a power constrain [22], or minimize weighted sum of mean response time and power consumption [5]. A survey of work on single server power cost reduction is given in [23]. For a data center, Gandhi et al. provide management policies that minimize mean response time under a total power cap [11] or minimize the product to response time and power cost [7]. Chen et al. propose solutions based on queuing models and control theory to minimize the server energy as well as data center operational costs [10]. Lin et al. design a three-competitive online algorithm to minimize a convex function of power usage and delay [4]. SAVE differs from these work in three ways: 1) it leverages spatio *and* temporal differences in job arrivals and power prices to achieve power cost reduction; 2) all work mentioned above except [22] and [21] assume closed-form convex expressions for service delay or delay-cost, whereas SAVE does not make these assumptions as they may not always hold, especially for delay tolerant jobs; and 3) SAVE does not rely on predictions of workload arrival processes, as [10] 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 6. Rao et al. provide an approximation algorithm for minimizing the total power cost subject to an average delay constraint [20], while Liu et al. designed load balancing algorithms to minimize a metric that combines power and delay costs [3]. Lin et al. [24] extend [4] to the case of multiple data centers with heterogeneous delay, power price, and server characteristics. All papers mentioned here make routing and server on/off decisions based on predictions on arrival rates and closed-form convex expressions for average delay. Stanojevic and Shorten [6] make control decisions at server, data center, and interdata center level every time slot by solving a deterministic convex optimization problem. All these works have work-conserving schedulers 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.

A number of recent works introduce new aspects in better usage of power in data centers. For example, Urgaonkar et al. [25] take battery into consideration and show that it can be used to reduce cost on power. El-Sayed et al. [26] study cooling strategy in data centers. Liu et al. [27] consider server management together with cooling and usage of renewable energy. These works are orthogonal to ours in that approaches used by these works can be combined with ours to achieve lower power usage/cost for data centers.

The Lyapunov optimization technique that we use to design SAVE was first proposed in [8] for network stability problems. It was generalized in [13] for network utility optimization problems. Recently, Urgaonkar et al. used this technique to design an algorithm for job admission control, routing, and resource allocation in a virtualized data center [28]. 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 workload management for geographically distributed data centers.

8 Conclusions

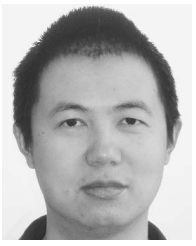
In this paper, we propose a general framework for power cost reduction in geographically distributed data centers. Our approach incorporates routing and server management actions on individual servers, within a data center, and across multiple data centers, and works at multiple time

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.

As noted earlier, our work considered a number of practical concerns. However, it still has several limitations, including: 1) instead of providing a detailed model for server processing delay, we simplified the problem by assuming that the processing time of each job is proportional to the amount of work it contains; 2) in our simulations, the workloads we created did not include dependences among tasks, or "stragglers," that are prevalent in real-world MapReduce workloads [17]; and 3) SAVE requires simultaneous activation/deactivation of multiple servers, which can potentially result in instability of the power grid. Part of our future efforts is to explore these issues. Other future work includes taking into consideration additional design aspects, such as renewable energy.

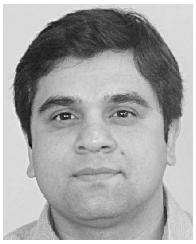
REFERENCES

[1] www.gizmodo.com/5517041/, 2013.
[2] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the Electric Bill for Internet-Scale Systems," *Proc. ACM SIGCOMM Conf. Data Communication (SIGCOMM '09),* 2009.
[3] Z. Liu, A. Wierman, S. Low, and L. Andrew, "Greening Geographical Load Balancing," *Proc. ACM SIGMETRICS Joint Int'l Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '11),* 2011.
[4] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic Right-Sizing for Power-Proportional Data Centers," *Proc. IEEE INFOCOM,* 2011.
[5] A. Wierman, L. Andrew, and A. Tang, "Power-Aware Speed Scaling in Processor Sharing Systems," *Proc. IEEE INFOCOM,* 2009.
[6] R. Stanojevic and R. Shorten, "Distributed Dynamic Speed Scaling," *Proc. 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," *Performance Evaluation,* vol. 67, pp. 1155-1171, Nov. 2010.
[8] L. Tassiulas and A. Ephremides, "Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks," *IEEE Trans. Automatic Control,* vol. 37, no. 12, pp. 1936-1949, Dec. 1992.
[9] A.K. Mishra, J.L. Hellerstein, W. Cirne, and C.R. Das, "Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters," *SIGMETRICS Performance Evaluation Rev.,* vol. 37, no. 4, pp. 34-41, Mar. 2010.
[10] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing Server Energy and Operational Costs in Hosting Centers," *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '05),* 2005.
[11] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal Power Allocation in Server Farms," *Proc. 11th Int'l Joint Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '09),* 2009.
[12] http://googleenterprise.blogspot.com/2010/03/disaster-recovery-by-google.html, 2013.
[13] L. Georgiadis, M.J. Neely, and L. Tassiulas, "Resource Allocation and Cross-Layer Control in Wireless Networks," *Foundations and Trends in Networking,* vol. 1, no. 1, pp. 1-149, 2006.
[14] L. Huang and M. Neely, "Max-Weight Achieves the Exact $[o(1/v), o(v)]$ Utility-Delay Tradeoff under Markov Dynamics," arXiv:1008.0200v1.
[15] Federal Energy Regulatory Commission, www.ferc.gov, 2013.
[16] 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," *Proc. Fifth European Conf. Computer systems (EuroSys '10),* 2010.
[17] 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," *Proc. Ninth USENIX Conf. Operating Systems Design and Implementation (OSDI '10),* 2010.
[18] www.google.com/corporate/green/, 2013.
[19] Unused Servers Survey Results Analysis, http://www.thegreengrid.org/, 2013.
[20] L. Rao, X. Liu, L. Xie, and W. Liu, "Minimizing Electricity Cost: Optimization of Distributed Internet Data Centers in a Multi-Electricity-Market Environment," *Proc. IEEE INFOCOM,* 2010.
[21] F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy," *Proc. 36th Ann. Symp. Foundations of Computer Science,* 1995.
[22] K. Pruhs, P. Uthaisombut, and G. Woeginger, "Getting the Best Response for your Erg," *ACM Trans. Algorithms,* vol. 4, pp. 1-17, July 2008.
[23] S. Albers, "Energy-Efficient Algorithms," *Comm. ACM,* vol. 53, pp. 86-96, May 2010.
[24] M. Lin, Z. Liu, A. Wierman, and L.L.H. Andrew, "Online Algorithms for Geographical Load Balancing," *Proc. Int'l Green Computing Conference (IGCC '12),* 2012.
[25] R. Urgaonkar, B. Urgaonkar, M.J. Neely, and A. Sivasubramaniam, "Optimal Power Cost Management Using Stored Energy in Data Centers," *Proc. ACM SIGMETRICS Joint Int'l Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '11),* 2011.
[26] N. El-Sayed, I.A. Stefanovici, G. Amvrosiadis, A.A. Hwang, and B. Schroeder, "Temperature Management in Data Centers: Why Some (Might) Like It Hot," *Proc. ACM SIGMETRICS,* 2012.
[27] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser, "Renewable and Cooling Aware Workload Management for Sustainable Data Centers," *Proc. 12th ACM SIGMETRICS/PERFORMANCE Joint Int'l Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '12),* 2012.
[28] R. Urgaonkar, U. Kozat, K. Igarashi, and M. Neely, "Dynamic Resource Allocation and Power Management in Virtualized Data Centers," *Proc. IEEE Network Operations and Management Symp. (NOMS),* 2010.



Yuan Yao received both the Bachelor of Science degree in computer science and the Bachelor of Science degree in measurement and control technique from the Huazhong University of Science and Technology, Wuhan, China, in 2004, and the PhD degree from the Department of Electrical Engineering Systems, University of Southern California, Los Angeles, in 2012. His research interests include performance modeling, analysis and design of distributed computing systems and applications.

Longbo Huang received the BE degree from Sun Yat-sen University, Guangzhou, China, in June 2003, the MS degree from Columbia University, New York, in December 2004, and the PhD degree from the University of Southern California, Los Angeles, in August 2011, all in electrical engineering. He then worked as a postdoctoral researcher in the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley from July 2011 to August 2012. He is currently an assistant professor in the Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China. His research interests include the areas of stochastic network optimization, network coding and delay, data center networking, and green IT.



Abhishek B. Sharma received the five-year Integrated MTech degree in mathematics and computing from the Indian Institute of Technology, Delhi, India, in 2003, and the PhD degree from the University of Southern California, Los Angeles. He is currently a research staff member at the NEC Labs, America, Princeton, New Jersey. His research interests include performance analysis, optimization, and autonomous decision making in large distributed systems such as sensor networks and compute clouds.

Leana Golubchik received the PhD degree from University of California, Los Angeles, where she is currently a professor in the Department of Computer Science, with a joint appointment in electrical engineering. Prior to that, she was an assistant professor and then an associate professor at the University of Maryland at College Park and an assistant professor at Columbia University. She is a past chair, vice chair, and board of directors member of ACM SIGMETRICS, and is the current editor of the *ACM Performance Evaluation Review.* She has served as a program cochair and general chair of ACM SIGMETRICS, program cochair of MIS, a cochair of HotMetrics, and is also on the editorial board of the *Performance Evaluation* journal. She received several awards, including the IBM Faculty Award, the NSF CAREER Award, the Okawa Foundation Award, and the IBM and NSF Doctoral Fellowships. She is a member of the IFIP WG 7.3 and Tau Beta Pi.

Michael J. Neely received the BS degrees in both electrical engineering and mathematics from the University of Maryland, College Park, in 1997. He was then awarded a Department of Defense NDSEG Fellowship for graduate study at the Massachusetts Institute of Technology, where he received the MS degree in 1999 and the PhD degree in 2003, both in electrical engineering. In 2004, he joined the faculty of the Electrical Engineering Department, University of Southern California, where he is currently an associate professor. His research interests include the area of stochastic network optimization for wireless and ad hoc mobile networks. He received the NSF Career Award in January 2008.