Distributed Resource Management in Data Center with Temperature Constraint

Mohammad A. Islam, Shaolei Ren, Niki Pissinou, Hasan Mahmud
Florida International University
Miami, FL 33199, United States
{misla012, sren, pissinou, amahm008}@fiu.edu

Athanasios V. Vasilakos
National Technical University of Athens
Athens 106 80, Greece
vasilako@ath.forthnet.gr

***Abstract*—With the ever-increasing power density generating an excessive amount of heat that potentially induces server damage and outages, data center operation must be judiciously optimized to prevent server overheating. Moreover, data center is subject to various peak power constraints (such as peak server power) that have to be satisfied at *all* times for reliability concerns. In this paper, we propose a novel resource management algorithm, called DREAM (Distributed REsource mAnagement with teMperature constraint), to optimally control the server capacity provisioning (via power adjustment) and load distribution for minimizing the data center *operational* cost while satisfying the IT peak power and maximum server inlet temperature constraints. By using DREAM, each server can autonomously adjust its *discrete* processing speed (and hence, power consumption, too) and optimally decide the amount of workloads to process, in order to minimize the total cost that incorporates both power consumption and service delay. We rigorously prove that DREAM can yield the minimum cost with an *arbitrarily* high probability while satisfying the peak power and inlet temperature constraints. To complement the analysis, we perform a simulation study and show that DREAM can significantly reduce the cost compared to the optimal temperature-unaware algorithm (by up to 23%) and equal load distribution (by up to 69%).**

***Index Terms*—Capacity provisioning, data center, distributed optimization, load distribution, temperature constraint.**

I. Introduction

In recent years, we have witnessed the emergence of a plethora of on-line applications and services, such as video sharing, social networking and e-commerce. As a consequence, together with the growing adoption of cloud computing, more and more data centers are required as a critical computing infrastructure, each consisting of tens of thousands of servers that consume an enormous amount of power. Meanwhile, data center operators have been increasingly pressured to deliver premium Quality of Service (QoS) and to reduce their power consumption for less carbon footprint.

Consuming less power without compromising QoS is appealing yet challenging, especially in data centers processing delay-sensitive workloads such as web services. While data center management have undergone a significant improvement [11], the following three challenges still widely exist, posing practical restrictions on data center operation. First, data centers have a stringent IT peak power budget: exceeding the peak power budget will result in serious consequences such as service outages and equipment damages [5, 6, 14]. Increasing the power budget, however, may not be a viable solution in practice, as the capital cost of building a data center is directly proportional to the provisioned IT peak power (currently, estimated at 10-20 U.S. dollars per Watt) [6]. Thus, judiciously allocating the total power budget to different server units while considering the peak power constraint is crucial for optimizing the data center operation. Second, with the ever-increasing power density generating an excessive amount of heat, thermal management in data centers is becoming imperatively important for preventing server overheating that could potentially induce server damages and huge economic losses [3]. As a consequence, optimally distributing the workloads to facilitate heat recirculation and avoid overheating has to be incorporated in data center operation. Last but not least, for system scalability, distributed resource management in data centers is highly desired, which, however, is not readily available in all scenarios.

In this paper, we propose a novel resource management algorithm, called DREAM (Distributed REsource mAnagement with teMperature constraint), to control the server capacity provisioning (via power adjustment) and load distribution for minimizing the data center *operational* cost while satisfying the IT peak power and maximum server inlet temperature[1] constraints. Unlike temperature-*reactive* approaches that prevent server overheating based on the observed real-time temperature (e.g., shut down servers when they become *hot* [3, 18]), DREAM makes distributed decisions while proactively taking into account the potential impact of the decisions on the inlet temperature increase to avoid server overheating. In our study, server power, cooling system power, and service delay are organically integrated as our optimization objective, thereby striking a flexible tradeoff between the power consumption and delay performance. By using DREAM, each server can autonomously adjust its *discrete* server processing speed (and hence, power consumption, too) and optimally decide the amount of workloads to process, in order to minimize the total cost (defined as a weighted sum of power consumption and delay cost). DREAM builds upon a variation of Gibbs sampling technique [28], combined with dual decomposition [2]. We

[1]Server inlet temperature is the temperature of air entering the server, and it is different from the server component temperature which is handled using a separate mechanism by the server itself (e.g., fan speed increases if the CPU temperature increases).

978-1-4799-0623-9/13/$31.00 ©2013 IEEE

conduct a rigorous performance analysis and formally prove that DREAM can yield the minimum cost, while satisfying the peak power and inlet temperature constraints. We also perform an extensive simulation study to complement the analysis. The simulation results are consistent with our theoretical analysis. Moreover, we compare DREAM with two existing algorithms and show that DREAM reduces the cost by more than 23%.

Our main contributions are summarized as follows.

1) We develop a distributed algorithm, DREAM, for data centers in which each server can autonomously decide its processing speed and the amount of workloads it needs to process, while incorporating three important practical constraints: limited processing speeds, peak server power, and maximum server inlet temperature constraints. It is rigorously proved that DREAM minimizes the total cost with an arbitrarily high probability.
2) We conduct a comprehensive simulation, and the results show that DREAM achieves a significantly lower cost compared to two widely-used existing algorithms while satisfying peak power and inlet temperature constraints.

The rest of this paper is organized as follows. Related work is reviewed in Section II, and the model is described in Section III. In Section IV and Section V, we present the problem formulation and develop our distributed online algorithm, DREAM, respectively. Section VI provides a simulation study to validate DREAM, and finally, concluding remarks are offered in Section VII.

II. Related Work

We provide a snapshot of the related work from the following aspects.

Data center optimization. There has been a growing interest in optimizing data center operation from various perspectives such as cutting electricity bills [8, 16, 24, 26, 27] and minimizing response times [6, 15]. For example, "power proportionality" via dynamically turning on/off servers based on the workloads (a.k.a. dynamic capacity provisioning or right-sizing) has been extensively studied and advocated as a promising approach to reduce the energy cost of data centers [8]. By exploring the geographical and temporal diversity of electricity prices, [24, 27] study geographical load balancing among multiple data centers to minimize energy cost. Combined with dynamic capacity provisioning, it has been shown that geographical load balancing can potentially provide additional benefits in cost saving [26], response time reduction [15], and reducing the brown energy usage by scheduling the workloads to data centers with more green energies [16]. In addition to energy cost reduction, minimizing the response time (i.e., maximizing the performance) is another important optimization objective for data centers. In [6], the peak power budget is optimally allocated to (homogeneous) servers to minimize the total response time based on a queueing-theoretic model; [14] studies a similar problem but in the context of virtualized systems; the research [15, 16] combines energy cost with response time by considering a parameterized optimization objective (as in our study). These studies assume server processing speed can be *continuously* chosen, which may not be practically realizable due to hardware constraints. Moreover, none of them have considered the temperature constraint (i.e., temperature-oblivious) or distributed resource management.

Temperature-aware resource management. Temperature-aware (or thermal-aware) resource management in data centers has recently attracted much research interest. In general, temperature-aware resource management can be classified as temperature-*reactive* and temperature-*proactive* approaches. In temperature-reactive approaches, decisions are made *reactively* to avoid server overheating based on the observed real-time temperature. For example, [18] dynamically schedules workloads to "cool" servers based on the instantaneously observed temperature to avoid server overheating; [13] optimally allocates power in a data center for MapReduce workloads by considering the impacts of real-time temperature on the server performance; [3] presents a cyber-physical approach for monitoring the real-time temperature distribution in data centers, facilitating temperature-reactive decisions. Unlike temperature-reactive approaches, temperature-proactive approaches explicitly take into account the impact of resource management decisions on the temperature increase. Nonetheless, directly using computational fluid dynamics (CFD) models for temperature prediction incurs a prohibitive complexity [1], leading to the development of a less complex yet sufficiently accurate heat transfer model, as shown in (11). For example, without considering delays, [19] maximizes the total capacity of a data center subject to temperature constraint based on this model; [20, 21, 30] develops software/hardware architectures for various types of thermal management in high-performance computing environments, e.g., minimizing the peak inlet temperature through task assignment to alleviate the cooling requirement. Our study, by contrast, considers discrete choices of service rates as well as delay costs (that are important for delay-sensitive workloads such as web services).

Gibbs sampling. The proposed DREAM is developed based on a variation of Gibbs sampling [28] combined with dual decomposition [2]. To our best knowledge, our study is the first to apply such technique for solving distributed processing speed selection and load distribution in data centers, although Gibbs sampling and dual decomposition has been recently used to solve distributed combinatorial optimization in engineering disciplines (e.g., power control in wireless networks [22, 25, 29]).

To summarize, unlike the existing research, we propose a distributed resource management algorithm, DREAM, using which each server can autonomously make (discrete) service rate and load distribution decisions while proactively taking into account the impact of their decisions on the inlet temperature increase to avoid server overheating.

III. Model

We consider a model in which the capacity provisioning and load distribution decisions are updated periodically and the period is sufficiently large (e.g., one hour) such that the

TABLE I
LIST OF NOTATIONS.

Notation	Description
x_i	Service rate of server i
λ_i	Workloads distributed server i
p_i	Average power consumption of server i
p^c	Cooling system power consumption
p	Total power consumption
d_i	Delay cost in server i
g	Total cost
T_{sup}	Supply temperature
$\mathbf{D}$	Heat transfer matrix
$\mathbf{T}_{\text{in}}$	Server inlet temperature

room temperature can become stabilized during each period. Throughout the paper, we drop the time index wherever applicable without affecting the analysis. Next, we present the modeling details for the data center and workloads. Key notations are summarized in Table I.

A. Data center

We consider a data center that has N servers that are mounted in M racks (or chassis). The m-th rack contains n_m servers such that $\sum_{m=1}^{M} n_m = N$. We denote the entire set of servers and the subset of servers mounted in the m-th rack by $\mathcal{N} = \{1, 2, \cdots, N\}$ and $\mathcal{N}_m$, respectively, and it follows naturally that $\mathcal{N} = \mathcal{N}_1 \bigcup \mathcal{N}_2 \bigcup \cdots \bigcup \mathcal{N}_M$ and $\mathcal{N}_i \bigcap \mathcal{N}_j = \varnothing$ if $i \neq j$. In general, the servers are heterogeneous in their power consumptions and processing speeds due to various reasons such as different purchase dates. Moreover, each server may trade performance for power consumption by varying its performance and power states (e.g., P-states, C-states, or a combination of them), varying its processing speed (e.g., via dynamic voltage and frequency scaling or DVFS [17]), or switching servers on/off. Note that the processing speed is quantified in terms of the *service rate* rather than the actual clock rate, i.e., how many jobs can be processed in a unit time (e.g., a second or an hour), and we interchangeably use "processing speed" and "service rate" wherever applicable. We also use "capacity" to represent the service rate of a server. To keep our model general, we consider that server i can choose its speed x_i out of a finite set $\mathcal{S}_i = \{s_{i,0}, s_{i,1}, \cdots, s_{i,K_i}\}$, where $s_{i,0} = 0$ represents zero speed (e.g., deep sleep or shut down) and K_i is the total number of positive processing speeds available to server i. Assuming that the servers consume a negligible power under the zero-speed mode, we express the average power consumption of server i as

$$p_i(\lambda_i, x_i) = \left[p_{i,s} + p_{i,c}(x_i) \cdot \frac{\lambda_i}{x_i} \right] \cdot \mathbf{1}_{(x_i > 0)}, \tag{1}$$

where λ_i is the workload arrival rate distributed to server i, $p_{i,s}$ is the static power regardless of the workloads as long as server i is turned on, $p_{i,c}(x_i)$ is the computing power incurred only when server i is processing workloads at a speed of x_i, and the indicator function $\mathbf{1}_{(x_i>0)} = 1$ if and only if the processing speed $x_i > 0$.

In our study, we only focus on the cooling power and server power for the considered workloads, while neglecting the power consumption of other parts (e.g., networking, power supply system) which however can be conveniently absorbed by a power usage effectiveness (PUE) factor [16]. In a data center, a large portion of the server power consumption translates into heat that poses potential risks of service disruption if not promptly exhausted [21]. Typically, as illustrated in Fig. 1, a computer room air conditioning (CRAC) unit blows cold air into the data center and exhaust air ducts are placed above the servers to remove the hot air. Due to the fast growing power density, cooling system incurs a significant power consumption in data centers (e.g., over 30% of the total power consumption) [21]. While it depends on several factors (e.g., air conditioner efficiency), the cooling power consumption can be approximated as a function of the supply temperature and the server power consumption [20, 21] given by

$$p^c = \frac{\sum_{i=1}^{N} p_i(\lambda_i, x_i)}{\text{CoP}(T_{\text{sup}})}, \tag{2}$$

where $\text{CoP}(T_{\text{sup}})$ is referred to as *coefficient of performance* of the cooling system given a temperature of T_{sup} for the supplied cold air. The coefficient of performance $\text{CoP}(T_{\text{sup}})$ characterizes the efficiency of the cooling system and, as corroborated by prior work that performs extensive simulation and modeling based on CFD [18, 20, 21], we can express $\text{CoP}(T_{\text{sup}})$ as follows

$$\text{CoP}(T_{\text{sup}}) = 0.0068 T_{\text{sup}}^2 + 0.0008 T_{\text{sup}} + 0.458. \tag{3}$$

Combining the cooling power and server power (but excluding the load distributor's power consumption which is negligible compared to the total power), the total power consumption[2] is given by

$$p(\vec{\lambda}, \vec{\mathbf{x}}) = \sum_{i=1}^{N} p_i(\lambda_i, x_i) \cdot \left[1 + \frac{1}{\text{CoP}(T_{\text{sup}})} \right], \tag{4}$$

where $\vec{\lambda} = (\lambda_1, \lambda_2, \cdots, \lambda_N)$ and $\vec{\mathbf{x}} = (x_1, x_2, \cdots, x_N)$ are the load distribution and capacity provisioning decisions, respectively. Our study can easily capture the electricity cost by multiplying (4) with the electricity price. Note that, since the decisions are updated infrequently (i.e., hourly in our study) as in [16, 26], we ignore the possible *toggling* costs incurred when changing the capacity provisioning decisions (e.g., turning a server off or into deep sleep) that can be dealt with using techniques as developed in [15].

B. Workloads

We denote by $\lambda \in [0, \lambda_{\max}]$ the total arrival rate of workloads in the data center, where $\lambda_{\max}$ is the maximum possible arrival rate. As assumed in prior work [15, 16], the value of λ is accurately available at the beginning of each decision period (e.g., by using hour-ahead workload prediction). In our study, we focus on delay-sensitive interactive workloads as in [6, 16, 26], whereas delay-tolerant batch workloads can

[2]This is equivalent to energy consumption, since the length of each decision period is the same.

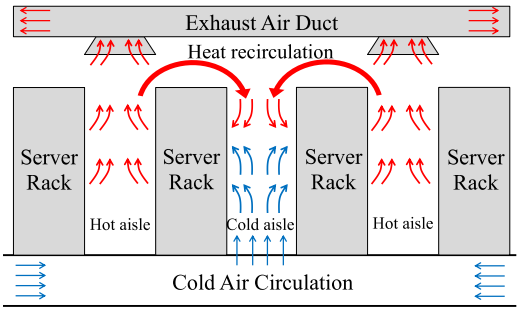


Fig. 1. System diagram

be easily captured (e.g., by maintaining a separate batch job queue) as considered by several existing studies [27]. The workloads first arrive at a load distributor before they are distributed to servers for processing. We denote the workload arrival rate distributed to server i by $\lambda_i \geq 0$.

We denote the *delay cost* for workloads at server i by a strictly convex function $d_i(\lambda_i, x_i)$, which is intuitively increasing in λ_i and decreasing in x_i [15, 16]. As a concrete example, we model the service process at each server as an M/M/1 queue and use the average response time (multiplied by the arrival rate) to represent the delay cost. Specifically, it is well known that the average response time for the M/M/1 queue at server i is $\frac{1}{x_i - \lambda_i}$ [24] and hence, the total delay cost can be written as

$$d(\vec{\lambda}, \vec{\mathbf{x}}) = \sum_{i=1}^{N} d_i(\lambda_i, x_i) = \sum_{i=1}^{N} \frac{\lambda_i}{x_i - \lambda_i}, \tag{5}$$

in which we ignore the network delay cost between the load distributor and servers. Note that the average network delay between the users and the data center can be approximately modeled as a certain constant [16] and added into (5) without affecting our approach of analysis. We are aware that M/M/1 queueing model may not capture the exact response time in practice because: (1) workload service time often follows a heavy-tailed distribution such as Pareto distribution [10]; and (2) the inter-arrival time may not be exponentially distributed [23]. However, as queuing models with non-exponentially distributed inter-arrival and service times are difficult to analyze, the M/M/1 model has been widely used as an analytic vehicle to provide a reasonable approximation for the actual service process [6, 16, 26]. In addition, our analysis is not restricted to the specific delay cost given by (5).

IV. Problem Formulation

In this section, we first specify the optimization objective and constraints for the data center operator, and then present the problem formulation for capacity provisioning and load distribution.

A. Objective and constraints

At the beginning of each decision period, the data center updates its capacity provisioning and load distribution decisions to minimize a parameterized cost subject to a set of constraints, as specified below.

Objective. We focus on operational costs rather than capital costs (e.g., building data centers, installing cooling systems). Both power consumption and delay cost are important for data centers, as the former takes up a dominant fraction of the operational cost while the later affects the user experience and revenue [15]. Our study incorporates both costs by considering a parameterized cost function as follows

$$g(\vec{\lambda}, \vec{\mathbf{x}}) = p(\vec{\lambda}, \vec{\mathbf{x}}) + \beta \cdot d(\vec{\lambda}, \vec{\mathbf{x}}), \tag{6}$$

where $p(\vec{\lambda}, \vec{\mathbf{x}})$ and $d(\vec{\lambda}, \vec{\mathbf{x}})$ are given by (4) and (5), respectively, and $\beta \geq 0$ is the weighting parameter adjusting the importance of delay cost relative to the (average) power consumption [15, 16]. In particular, if β reduces to zero, the data center only minimizes the power consumption, while if β approaches infinity, only the delay cost is minimized.

Constraints. The first natural constraint is that server i can only select one of its supported service rates, i.e.,

$$x_i \in \mathcal{S}_i = \{s_{i,0}, s_{i,1}, \cdots, s_{i,K_i}\}, \ \forall i. \tag{7}$$

Moreover, to avoid server overloading and workload dropping, the load distribution decisions need to satisfy

$$0 \leq \lambda_i \leq \gamma \cdot x_i, \ \forall i \tag{8}$$

$$\sum_{i=1}^{N} \lambda_i = \lambda, \tag{9}$$

where $\gamma \in (0, 1)$ is a predetermined parameter that controls the maximum utilization of a server. In a data center, multi-level peak power budgets (e.g., data center-level peak power [6], rack-level server power [14]) are strictly imposed for cost and reliability reasons. As in [6], we focus on the total peak server power constraint expressed as

$$\sum_{i=1}^{N} [p_{i,s} + p_{i,c}(x_i)] \leq \hat{P}, \tag{10}$$

where $p_{i,s} + p_{i,c}(x_i)$ is the peak power of server i running at a capacity of x_i. Other peak power constraints, e.g., rack-level peak power constraint, can also be easily added in our formulation.

Next, we specify the *maximum* temperature constraint as follows. While the server power consumption directly affects the amount of heat generated, the rise of temperature in hot aisles is typically a slow process (e.g., 10-15 minutes [21]) and thus, it depends on the average server power rather than the peak power consumption. As illustrated in Fig. 1, the generated heat recirculates in the data center (i.e., heated outlet air returns to the inlet of the servers), thereby imposing potential reliability issues for data center operation. Using CFD analysis, prior work [18, 20, 21] has shown that the relation between the average server power consumption and air exhaust inlet temperature is governed by the following equation

$$\mathbf{T}_{\text{in}} = \mathbf{T}_{\text{sup}} + \mathbf{D} \times \mathbf{p}, \tag{11}$$

where $\mathbf{T}_{\text{in}} = (T_{1,\text{in}}, T_{2,\text{in}}, \cdots, T_{M,\text{in}})'$ and $\mathbf{T}_{\text{sup}} = (T_{\text{sup}}, T_{\text{sup}}, \cdots, T_{\text{sup}})'$ are (column) vectors of inlet temperatures and supply temperatures for each rack, respectively, $\mathbf{D}$ is referred to as the *heat transfer matrix* converting the average server power consumption to temperature increases, and $\mathbf{p} = (\bar{p}_1, \bar{p}_2, \cdots, \bar{p}_M)'$ in which $\bar{p}_m = \sum_{i \in \mathcal{N}_m} p_i(\lambda_i, x_i)$ is the total average power consumption of servers mounted in the m-th rack. The heat transfer matrix $\mathbf{D}$ captures the spatial difference in temperature increases and the *heat recirculation* phenomena in data centers, and in particular, the element $D_{i,j}$ specifies the temperature increasing rate in the i-th rack caused by the servers in the j-th rack due to heat recirculation. To avoid server overheating and ensure the best performance of servers, a maximum inlet temperature constraint is imposed, i.e., the data center operation must satisfy the following constraint

$$\mathbf{T}_{\text{in}} \preceq \hat{\mathbf{T}}, \tag{12}$$

where $\hat{\mathbf{T}} = \{\hat{T}_1, \hat{T}_2, \cdots, \hat{T}_M\}$ specifies the maximum inlet temperature for each rack and "$\preceq$" is the element-wise inequality. In practice, the maximum temperature constraint is usually the same for each rack, i.e., $\hat{T}_1 = \hat{T}_2 = \cdots = \hat{T}_M = \hat{T}$.

B. Optimization problem

This subsection presents the optimization problem formulation for capacity provisioning and load distribution as follows:

$$\textbf{P1}: \quad \min_{\mathcal{A}} g(\vec{\lambda}, \vec{\mathbf{x}}) = p(\vec{\lambda}, \vec{\mathbf{x}}) + \beta \cdot d(\vec{\lambda}, \vec{\mathbf{x}}), \tag{13}$$

$$s.t., \quad \text{constraints } (7), (8), (9), (10), (12), \tag{14}$$

where $\mathcal{A}$ represents all the feasible capacity provisioning and load distribution decisions that we need to optimize. While incorporating real-time feedback from the environment (e.g., temperature monitoring result [3]) may help refine decisions over the course of operation, we note that it is orthogonal to our study, as our focus is on devising temperature-proactive approaches that explicitly incorporates the impact of data center decisions on the inlet temperature increase.

Throughout the paper, we assume that the problem **P1** is feasible, i.e., there exists at least one decision that satisfies all the constraints specified in (14). Note that, although our main focus is on optimizing capacity provisioning and load distribution decisions, we will also address the optimal choice of supply temperature T_{sup} that significantly affects the cooling power consumption, as can be seen from (2). While the optimization objective is clear, there exists a major challenge that impedes the derivation of the optimal solution to **P1**. Specifically, as can be seen from the constraint (7), **P1** belongs to mixed-integer nonlinear programming that is intrinsically difficult to solve (and even if solved in a centralized manner, the complexity will quickly exceed the computational capability of a single node when the number of servers increases) [2, 29]. Moreover, if the supply temperature T_{sup} is also included as an optimization variable, the problem becomes even more difficult. To address these challenges, we propose a distributed algorithm in Section V, in which each server autonomously makes capacity provisioning and load distribution decisions.

Algorithm 1 DREAM

1: Initialization: each server i chooses a feasible processing speed x_i^* and load distribution λ_i^*, for $i = 1, 2, \cdots, N$; set $\vec{\mathbf{x}} \leftarrow \vec{\mathbf{x}}^*$, $\vec{\lambda} \leftarrow \vec{\lambda}^*$, $\tilde{g} \leftarrow \infty$
2: **if** $\lambda \leq \gamma \cdot \sum_{i=1}^{N} x_i^*$ **then**
3: Obtain $\vec{\lambda}^*$ by solving

$$\min_{\vec{\lambda}} \left[p(\vec{\lambda}, \vec{\mathbf{x}}^*) + \beta \cdot d(\vec{\lambda}, \vec{\mathbf{x}}^*) \right], \tag{15}$$

subject to (8),(9), (12), and set $\tilde{g}^*$ to the minimum value of (15); set $\tilde{g}^* \leftarrow \infty$ if minimizing (15) subject to (8),(9),(12) is not feasible
4: $u \leftarrow \frac{\exp\left(\frac{\delta}{\tilde{g}^*}\right)}{\exp\left(\frac{\delta}{\tilde{g}}\right) + \exp\left(\frac{\delta}{\tilde{g}^*}\right)}$
5: With a probability of u: each server i sets $x_i \leftarrow x_i^*$, $\lambda_i \leftarrow \lambda_i^*$ and $\tilde{g} \leftarrow \tilde{g}^*$; with a probability of $1-u$: each server i sets $x_i^* \leftarrow x_i$
6: **end if**
7: Randomly select a server i; server i randomly selects a processing speed $x_i' \in \mathcal{S}_i$ and sets $x_i^* \leftarrow x_i'$
8: Return $\vec{\mathbf{x}}$ and $\vec{\lambda}$ if the stopping criterion is satisfied; otherwise, go to Line 2

V. Distributed Capacity Provisioning and Load Distribution

This section presents our proposed algorithm, DREAM, which can be implemented in a distributed manner: each server makes autonomous decisions. We also rigorously prove that DREAM yields the optimal solution with an *arbitrarily* high probability.

A. DREAM

As aforementioned, **P1** is an optimization problem involving mixed-integer nonlinear programming. While there exist various centralized techniques (such as Generalized Benders Decomposition [7]) to solve it, distributed solutions are desired such that each server can make autonomous decisions for system scalability. To solve **P1**, we propose a distributed algorithm based on a variation of Gibbs sampling [28, 31], which we refer to as DREAM, presented in Algorithm 1.

DREAM is a distributed algorithm working as follows: at each iteration, a randomly selected server first autonomously updates its processing speed, and then the servers decide their optimal load distribution decisions (also distributedly), after which the servers communicate decisions to each other. Alternatively, a coordinating node (e.g., load balancer) may facilitate message passing by collecting and distributing information exchanges, while in this case DREAM becomes *semi*-distributed. Line 3 in DREAM, i.e., minimizing (15), can be solved distributedly using dual decomposition [2], whose details are omitted due to space limitations but available in [12]. Note that during the iterations, servers do not need to actually adjust their speeds or load distribution decisions, which is only needed after the completion of the algorithm. In line 7, to randomly select a server, we can assign each server with a

random timer to "compete" for the updating opportunity, like in random channel access in wireless networks. In the event of server failures, only functioning servers need to participate in DREAM, while those failed servers do not intervene the execution of the algorithm.

Now, we explain why DREAM yields a better solution than greedy approaches. It is known that always choosing a better decision (i.e., greedy approach) may lead to an arbitrarily bad outcome for combinatorial optimization [28, 29]. To avoid the inefficiency, DREAM explores new solutions by *deliberately* introducing randomness to decision making (i.e., line 5), even though they may be worse than the current solution [31]. The degree of randomness is controlled by a smoothing parameter $\delta > 0$ that can be tuned to adjust the tradeoff between exploration and exploitation. On one hand, as δ becomes large, DREAM is more *greedy* and keeps a new solution with a greater probability if it is better than the current solution (i.e., $\tilde{g}^* \leq \tilde{g}$). In this case, however, it takes more iterations to identify the optimal solution because DREAM places more emphasis on exploitation and may be stuck in a locally optimal solution for a long time before successfully exploring other less greedy solutions that lead to more efficient outcomes. On the other hand, as $\delta \to 0$, DREAM tries more aggressively to explore all the possible solutions from time to time even though they are worse than the current one, and as a consequence, DREAM will quickly arrive at the optimal solution but then keep on oscillating without sticking to the optimal solution. Thus, in practice, it is advisable to gradually increase δ such that both exploration and exploitation are leveraged. Note that the randomness introduced in line 5 is a variation of Gibbs sampling [28, 31]: in line 5, only the old decision and the randomly selected new decision are *sampled* probabilistically, unlike the original Gibbs sampling technique that samples all the possible decisions and hence requires the servers to know the costs for all the possible decisions.

B. Analysis

In this subsection, we discuss the complexity and prove the performance of DREAM, and then we show how to extend DREAM to incorporate the supply temperature T_{sup} as a decision variable to further reduce the cost.

Complexity. The smoothing parameter $\delta > 0$ can be tuned to adjust the tradeoff between the computational complexity and the cost saving. When δ is large, the total complexity of DREAM may exceed that of the centralized exhaustive search (because DREAM is more likely to be trapped in a local optimal solution for a long time before exploring reaching the globally optimal solution), although in practice a reasonably good solution is often quickly identified. In practice, the computational complexity of DREAM can be effectively reduced by making capacity provisioning decisions on a *group* basis: changing speed selections for a whole group of servers in batch (analogous to the concept of *virtual machine pool* in large-scale virtualzed data centers [9]). We will illustrate in the next section the iteration process of DREAM under different values of δ.

Performance. Theorem 1 shows that DREAM can solve the optimization problem **P1** with an arbitrarily high probability.

Theorem 1. *As $\delta > 0$ increases, Algorithm 1 converges with a higher probability to the globally optimal solution that minimizes* (13) *subject to* (8),(9),(12). *When $\delta \to \infty$, Algorithm 1 converges to the globally optimal solution with a probability of 1.*

Proof: Due to page limit, we only provide an outline of the proof, while the details can be found in [12]. Theorem 1 can be established by showing that the iteration process based on Gibbs sampling follows a Markov chain, since at each iteration only one server updates its decision. The stationary points of the Markov chain are (locally) optimal solutions for minimizing (13) subject to (8),(9) and (12). In particular, we obtain the stationary probability distribution as

$$\Omega_{\vec{\mathbf{x}}} = \frac{\exp\left(\frac{\delta}{\tilde{g}(\vec{\mathbf{x}})}\right)}{\sum_{\vec{\mathbf{x}}'} \exp\left(\frac{\delta}{\tilde{g}(\vec{\mathbf{x}}')}\right)}, \tag{16}$$

where $\vec{\mathbf{x}}$ is the capacity provisioning vector and $\tilde{g}(\vec{\mathbf{x}})$ is the corresponding cost (in which we suppress the load distribution decision for brevity). Denote $\vec{\mathbf{x}}^*$ as the optimal solution, and suppose temporarily that the optimal decision is unique. By taking the first-order derivative of (16) with respect to δ, it can be seen that the probability of converging to $\vec{\mathbf{x}}^*$ increases with δ. By letting the smoothing parameter $\delta \to \infty$, we obtain

$$\lim_{\delta \to \infty} \Omega_{\vec{\mathbf{x}}} = \begin{cases} 1, & \text{if } \vec{\mathbf{x}} = \vec{\mathbf{x}}^*, \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

Note that if there exist $J \geq 1$ capacity provisioning decisions, all of which minimize $\tilde{g}(\vec{\mathbf{x}})$, then Algorithm 1 will converge to each of the optimal decisions with a probability of $\frac{1}{J}$, as $\delta \to \infty$. ■

Theorem 1 indicates that using Algorithm 1, the servers can select the optimal processing speeds with an arbitrarily high probability (at the expense of slow exploration, once DREAM is stuck in a locally optimal solution).[3] To avoid being trapped in a locally optimal solution for a long time and yet achieve a good performance, an advisory approach used in practice is selecting the smoothing parameter δ *adaptively*: a small δ is initially chosen to explore all possible decisions, whereas δ is increased over the iterations such that the servers progressively *concentrate* on better solutions.

Optimizing supply temperature. In the above analysis, the temperature of cold air supplied by the CRAC unit is treated as is. Nonetheless, the supply temperature significantly affects the cooling power consumption: within the normal operational temperature range, the higher supply temperature, the less cooling power [18, 20, 21]. Now, we extend DREAM to incorporate the supply temperature T_{sup} as an additional decision variable. Specifically, we denote by $\mathcal{T} = \{T_1, T_2, \cdots, T_N\}$ the (discrete) set of available supply temperatures that are

[3] A more rigorous analysis of convergence rate for the original Gibbs sampling technique in the context of wireless networks is available in [25].

TABLE II
POWER CONSUMPTION AND NORMALIZED SPEED.

Type-1 Server	**Speed**	0.55	0.72	0.83	1.00
	Power	205W	225W	245W	283W
Type-2 Server	**Speed**	0.50	0.61	0.78	0.89
	Power	200W	210W	233W	255W

available to the CRAC unit.[4] We let a coordinating server, e.g., load balancer, to autonomously decide the supply temperature $T_{\text{sup}} \in \mathcal{T}$: the coordinator participates in the random selection process (specified by Line 7 of Algorithm 1) and, if selected to update, the coordinator randomly explores a new supply temperature, while the remaining part of the algorithm proceeds following Algorithm 1. For brevity, we omit the description of the new algorithm as well as its proof of optimality.

VI. SIMULATION

This section presents trace-based simulation studies of a university data center to validate our analysis and evaluate the performance of DREAM. We first present our simulation settings and then show the following three sets of results:

- Execution of DREAM: We show the iteration process of executing DREAM with different starting points and smoothing factors.
- Comparison with existing algorithms: We compare DREAM with two existing algorithms and show that DREAM reduces the total cost by more than 23% while satisfying the temperature constraint.
- Sensitivity study: We investigate the impacts of temperature constraint, workload and inaccurate estimate of the heat transfer matrix $\mathbf{D}$ on the performance of DREAM.

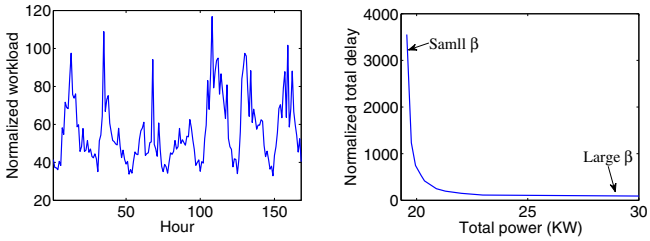
A. Settings

As in prior work (e.g., [18, 20, 21]), we conduct extensive simulations based on settings obtained by CFD analysis to validate DREAM, given the practical difficulties of running our experiments on a large data center. The capacity provisioning and load distribution decisions are updated hourly. The default settings are specified as follows and will be used throughout the simulations unless otherwise stated.

• Server: We simulate a small-scale data center with one CRAC unit and five racks installed in a row, each consisting of 48 servers. Note that a large data center normally consists of multiple smaller server rooms, each of which may have its own CRAC unit, and hence DREAM is still applicable by running it for resource management in *each* server room separately.

We consider two types of servers, each of which supports four different speeds via DVFS, and these two types of servers are mounted in rack #1,2,3 and rack #4,5, respectively. The service rate (or effective processing speed) of each server is normalized with respect to that of the most powerful server running at the maximum speed. The power consumption given each normalized service rate is selected based on the measurement in [4] and listed in Table II. The total peak server power constraint is 60KW.

[4]Continuous supply temperatures can be treated in a similar way [25].



(a) Workload trace (b) Impact of β

Fig. 2. Workload and impact of β

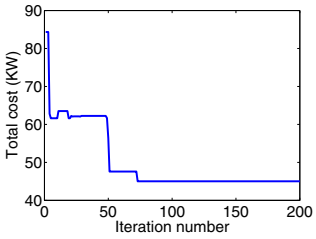
• Workloads: We profile the server usage log of Florida International University (FIU, a large public university in the U.S. with over 50,000 students) from December 1 to December 7, 2012. We scale the FIU workload arrival rates and show in Fig. 2(a) the normalized values with respect to the server capacity (i.e., arrival rate of 1 means it requires a service rate of at least 1).

• Others: Following [20, 21, 30], we set the 5×5 heat transfer matrix $\mathbf{D}$ as follows: $D_{i,j} = \frac{0.7^{|i-j|}}{1650}$, for $i, j = 1, 2, \cdots, 5$. By default, the supply temperature and maximum inlet temperature constraint are $10°\text{C}$ (unless optimized) and $25°\text{C}$, respectively. The unit of power consumption is watt. As the service rate is normalized, the average delay cost is also normalized and the weighting parameter β is 30, converting one unit of normalized delay cost into 30 watts of power consumption. Thus, the total cost (i.e., power plus weighted normalized delay) in our simulations has a unit of "W", although we use "KW" in the figures to scale down the values. The default smoothing parameter δ in Algorithm 1 is 5×10^6, which gives us a good performance at a reasonable convergence rate (see Fig. 3 for more details).

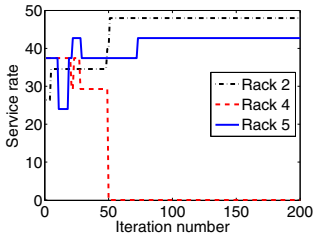
B. Execution of DREAM

We first show in Fig.2(b) the impact of β on the performance of DREAM. As intuitively expected, appropriately choosing β can achieve a flexible tradeoff between the delay performance and power consumption: with a larger β, the data center places more emphasis on the delay cost and thus, the delay cost is reduced whereas the power cost is increased.

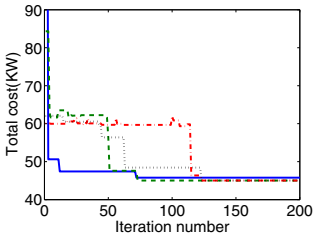
Next, we show the iteration process of executing DREAM in Fig. 3 under default settings and with a normalized arrival rate of 68. For illustration purposes, all the servers within one rack are assumed to choose the same service rate (i.e., one server chooses the service rate on behalf of the whole rack). Fig. 3(a) and Fig. 3(b) show the iteration of the cost $g(\vec{\lambda}, \vec{\mathbf{x}})$ and the iteration of the rack-level capacity provisioning (i.e., the sum capacity of a rack), respectively. For the convenience of presentation, only the capacities of rack #2,4,5 are shown in Fig. 3(b). Note that for the given workload, servers in rack #4 are shut down even though they have the same configuration as those in rack #5, because they are mounted closer to the servers in rack #1,2,3 and will result a more significant



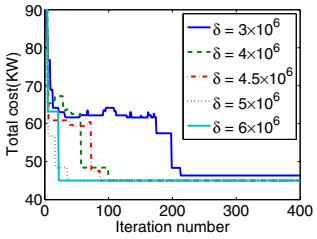
(a) Iteration of cost



(b) Iteration of capacity provisioning

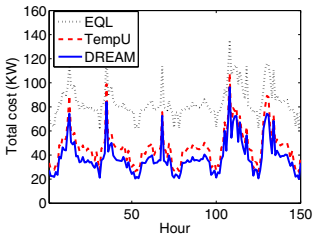


(c) Iteration with different initial points

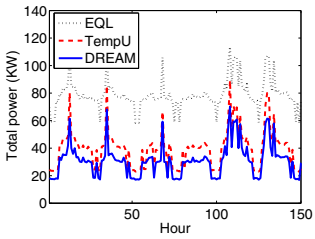


(d) Iteration with different δ

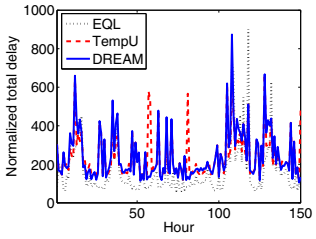
Fig. 3. Execution of DREAM



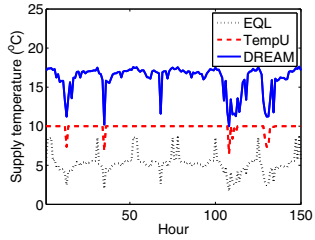
(a) Total cost



(b) Total average power



(c) Total delay cost



(d) Supply temperature

Fig. 4. Performance comparison

temperature increase due to heat transfer. We also observe that during the iterations, DREAM explores bad solutions that even *temporarily* increase the cost. Nonetheless, such exploration is essential to avoid being completely trapped in a locally optimal solution [29]. In Fig. 3(c), we show that with different initial points, the iterations of DREAM lead to almost the same outcome, thereby demonstrating the robustness of DREAM against the choices of initial points. Fig. 3(d) illustrates the iteration processes under different smoothing parameters δ and verifies our statement: with a smaller value of δ, DREAM is less greedy and explores new solutions more aggressively, whereas with a larger value of δ, DREAM is more greedy but achieves the globally optimal solution with a greater probability at the expense of slower exploration.

C. Comparison with existing algorithms

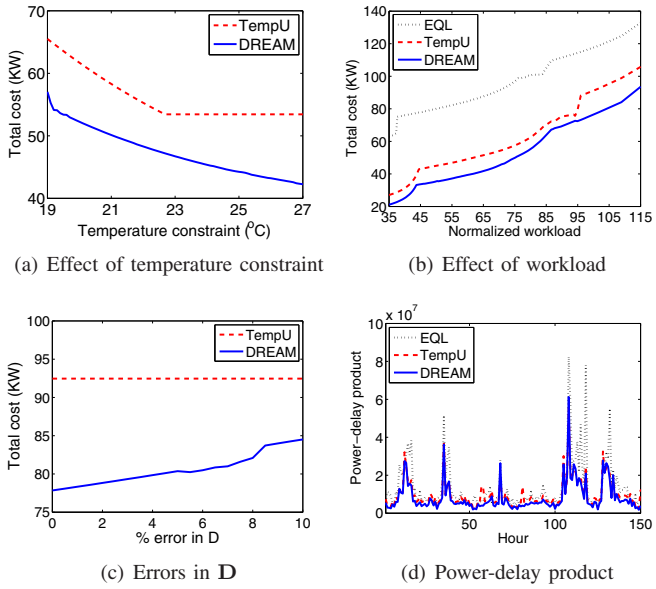
This subsection compares DREAM with two existing algorithms specified as follows.[5]

- EQL (Equal load distribution): Workloads are equally distributed across all the active servers. Service rates are selected in a greedy manner subject to the peak power constraint (i.e., the service rates are selected in a descending order until the peak power constraint is violated). Using real-time server temperature information, if a server is found to be overheated (i.e., its inlet temperature exceeds the constraint), it will be shut down; if the remaining servers cannot possibly process the workloads, a lower supply temperature will be applied.
- TempU (Temperature unaware): Capacity provisioning and load distribution decisions are made using Algorithm 1, except for that the maximum inlet temperature constraint is not explicitly taken into account during the optimization process.

[5]While our heat recirculation model follows [20], the scheduling algorithm in [20] is intended for HPC jobs and does not apply to our considered delay-sensitive workloads.

EQL is a variant of the temperature-reactive algorithm in [18] that schedules workloads to "cool" servers, while TempU represents the *best* temperature-oblivious algorithm. In both EQL and TempU, the default supply temperature is set as 10°C if the temperature constraint is satisfied, and it is reduced to an appropriate value otherwise. For DREAM, the supply temperature is optimized to minimize the cost, as discussed in Section V. While in principle the supply temperature could also be optimized (e.g., by exhaustive search) for EQL and TempU, we use the default supply temperature for these two algorithms because they are temperature-oblivious and in practice cannot choose *a priori* the optimal supply temperature, unless they violate the temperature constraint and the supply temperature has to be reduced *reactively*. On the other hand, DREAM adopts a proactive approach by explicitly taking into account the impact of capacity provisioning and load distribution decisions on the temperature increase, and thus the supply temperature can be optimally chosen in advance. Note that choosing other supply temperatures will not affect the general trend of our results as discussed in the next subsection.

We first show in Fig. 4 hourly comparison between DREAM and the two existing algorithms using our one-week workload trace. Specifically, Fig. 4(a) demonstrates that DREAM can significantly reduce the total cost compared to TempU (by up to 23%) and EQL (by up to 69%). We also observe that, compared to EQL, the cost reduction achieved by DREAM is more significant when the workload arrival rate is lower, because DREAM has more freedom to judiciously decide the service rates and load distribution whereas EQL only shuts down overheated servers and incurs a large idle power consumption. Fig. 4(b) shows that DREAM consumes the least average power, because: (1) DREAM can optimize the supply temperature in advance whereas TempU cannot; and (2) DREAM judiciously turns off some servers whereas EQL



(a) Effect of temperature constraint

(b) Effect of workload

(c) Errors in $\mathbf{D}$

(d) Power-delay product

Fig. 5. Sensitivity study

does not. Fig. 4(c) shows the total and average normalized delays, respectively. EQL incurs the smallest delay (except when many servers are shut down due to overheating), because EQL provides the highest service rate while DREAM and TempU places more emphasis on the power consumption (i.e., the weighing parameter β for the delay cost is relatively small). Meanwhile, DREAM and TempU achieves roughly the same delay: TempU makes similar decisions (except for the supply temperature) when the workload is light, because considering the temperature constraint will not make a significant impact on the decisions in the event of light workloads. Nonetheless, we notice from Fig. 4(c) that the delay achieved by using DREAM is significantly less than TempU for some hours. This is mainly because TempU chooses to turn on fewer servers to save power (as can be seen from Fig. 4(b)), thereby providing a lower service rate that may significantly increase the delay due to high utilization.

Next, we illustrate the hourly supply temperature in Fig. 4(d). It can be seen that, as EQL incurs the largest (server) power consumption, it requires the lowest supply temperature which, in turns, increases the cooling power and exaggerates the total power consumption. As mentioned above, the default supply temperature for TempU can already satisfy the temperature constraint under light workloads and hence, it only needs to be lowered under heavy workloads. DREAM can optimally adjust the supply temperature in response to the workloads: a higher supply temperature can be chosen to save the cooling power consumption when the workload arrival rate is lower.

D. Sensitivity study

In this subsection, we present the results of sensitivity studies by varying the temperature constraint, workload arrival rate, errors in the heat transfer matrix $\mathbf{D}$ and the performance metric. We only show the comparison in terms of the total cost due to space limitations, while noting that the power and delay comparisons are similar and that DREAM can be easily tuned by adjusting the weighting parameter β to achieve a flexible tradeoff between power and delay.

Effect of temperature constraint. To assess the robustness of DREAM in terms of the temperature constraint, we show in Fig. 5(a) the effect of the maximum inlet temperature constraint[6] on the total cost under a workload of 70. EQL is not shown in Fig. 5(a), as its cost exceeds 90 even though the temperature constraint increases to $27°\mathrm{C}$. Fig. 5(a) illustrates that as the temperature constraint increases, the total cost decreases (mainly because the cooling system can supply *hotter* air and incurs less power). Nonetheless, the cost of TempU becomes constant when the temperature constraint exceeds approximately $23°\mathrm{C}$, because the default temperature of $10°\mathrm{C}$ is low enough to satisfy the temperature constraint and TempU cannot choose the optimal supply temperature in advance due to its temperature-oblivious nature. Even though the supply temperature is optimized for TempU (e.g., when the temperature constraint is lower than $23°\mathrm{C}$), DREAM still achieves a lower cost than TempU, as it takes a proactive approach to avoid server overheating: DREAM explicitly considers the impact of capacity provisioning and load distribution decisions on the inlet temperature increase.

Effect of workload. We vary the workload to show its effect on the cost in Fig. 5(b) under the default maximum inlet temperature constraint of $25°\mathrm{C}$. It can be seen that DREAM achieves a lower cost than TempU under all the workloads, especially when the workload is high: with high workloads, TempU is forced to lower the supply temperature (which increases the cooling power), whereas DREAM can proactively schedule workloads to avoid server overheating while maintaining a relatively high supply temperature (as corroborated by Fig. 4(d)). EQL incurs the highest cost, because it only reactively shuts down overheated servers and incurs a large energy consumption.

Errors in $\mathbf{D}$. The heat transfer matrix $\mathbf{D}$ is crucial for DREAM to make optimal decisions to avoid sever overheating. In practice, $\mathbf{D}$ is derived by measurement combined with CFD analysis [20, 21, 30], and thus, it may not be *accurately* obtained. To address the errors in $\mathbf{D}$, we use a conservative approach: we use the worst-case conservative heat transfer matrix during the optimization. Specifically, during the optimization, we *assume* a new heat transfer matrix that is generated by increasing all the elements $D_{i,j}$ by a factor of error margins (e.g., 5% error margin means that each $D_{i,j}$ is increased by 5%). Fig. 5(c) shows the result under a relatively high workload arrival rate of 100; DREAM achieves a lower cost than TempU even though there are 10% errors in obtaining the heat transfer matrix $\mathbf{D}$. TempU ignores the temperature constraint during its optimization and hence achieves a constant cost regardless of the errors in $\mathbf{D}$.

[6]This is essentially equivalent to changing the default supply temperature for EQL and TempU (except for that it results in a different cooling power consumption).

Power-delay product. In Fig. 5(d), we compare DREAM with EQL and TempU under another important performance metric: power-delay product (equivalent to energy-delay product that is widely used in the scheduling literature [32], as the length of each decision period is the same in our study). Note that in the power-delay product, we use the total normalized delay, which is the average delay multiplied by the given workload arrival rate. While in general EQL incurs a lower delay (except when many servers are shut down due to overheating), it also incurs higher energy consumption. Thus, although minimizing the power-delay product is not the objective of DREAM, we see that DREAM still outperforms EQL as well as TempU in terms of the power-delay product, making DREAM even more appealing in practice.

Finally, we note that we also performed other sensitivity studies (e.g., different heat transfer matrixes, workload arrival rate estimation errors, server toggling costs) and found that the results are similar to the above, thereby demonstrating the effectiveness of DREAM in different systems and environments.

VII. Conclusion

We proposed a resource management algorithm, called DREAM, to optimally control the server capacity provisioning and load distribution for minimizing the data center *operational* cost that incorporates both energy consumption and service delay. By using DREAM, each server can autonomously adjust the *discrete* server processing speed (and hence, power consumption, too) and optimally decide the amount of workloads to process. We conducted a rigorous performance analysis and formally proved that DREAM can yield the minimum cost, while satisfying the peak power and inlet temperature constraints. We also performed an extensive simulation study to validate the analysis: we compared DREAM with two existing algorithms and showed that DREAM reduces the cost by more than 23%.

References

[1] A. H. Beitelmal and C. D. Patel. Thermo-fluids provisioning of a high performance high density data center. *Distrib. Parallel Databases*, 21(2-3):227–238, June 2007.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1989.

[3] J. Chen, R. Tan, Y. Wang, G. Xing, X. Wang, X.-D. Wang, B. Punch, and D. Colbry. A high-fidelity temperature distribution forecasting system for data centers. In *RTSS*, 2012.

[4] X. Chen, X. Liu, S. Wang, and X.-W. Chang. Tailcon: Power-minimizing tail percentile control of response time in server clusters. In *SRDS*, 2012.

[5] J. Choi, S. Govindan, B. Urgaonkar, and A. Sivasubramaniam. Profiling, prediction, and capping of power consumption in consolidated environments. MASCOTS, 2008.

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

[7] A. M. Geoffrion. Generalized benders decomposition. *J. of Optimization Theory and Applications*, 10(4):237–260, 1972.

[8] B. Guenter, N. Jain, and C. Williams. Managing cost, performance and reliability tradeoffs for energy-aware server provisioning. In *IEEE Infocom*, 2011.

[9] A. Gulati, G. Shanmuganathan, A. Holler, and I. Ahmad. Cloud-scale resource management: challenges and techniques. In *HotCloud*, 2011.

[10] M. Harchol-Balter. Task assignment with unknown duration. *J. of ACM*, 49(2):260–288, 2002.

[11] B. Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, July 2008.

[12] M. A. Islam, S. Ren, N. Pissinou, H. Mahmud, and A. Vasilakos. Distributed resource management in data centers with temperature constraint. *Tech. Report*, http://www.cs.fiu.edu/~sren/doc/tech/igcc13full.pdf.

[13] S. Li, T. Abdelzaher, and M. Yuan. Tapa: Temperature aware power allocation in data center with map-reduce. In *IGCC*, 2011.

[14] H. Lim, A. Kansal, and J. Liu. Power budgeting for virtualized data centers. In *USENIX ATC*, 2011.

[15] M. Lin, Z. Liu, A. Wierman, and L. L. H. Andrew. Online algorithms for geographical load balancing. In *IGCC*, 2012.

[16] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew. Greening geographical load balancing. In *SIGMETRICS*, 2011.

[17] J. R. Lorch and A. J. Smith. Pace: A new approach to dynamic voltage scaling. *IEEE Trans. Computers*, 53:856–869, July 2004.

[18] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *USENIX ATEC*, 2005.

[19] K. Mukherjee, S. Khuller, and A. Deshpande. Saving on cooling: the thermal scheduling problem. In *SIGMETRICS*, 2012.

[20] T. Mukherjee, A. Banerjee, G. Varsamopoulos, S. K. S. Gupta, and S. Rungta. Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers. *Comput. Netw.*, 53(17):2888–2904, Dec. 2009.

[21] T. Mukherjee, Q. Tang, C. Ziesman, S. K. S. Gupta, and P. Cayton. Software architecture for dynamic thermal management in datacenters. In *COMSWARE*, 2007.

[22] D. P. Palomar and M. Chiang. Alternative distributed algorithms for network utility maximization: Framework and applications. *IEEE Trans. Automatic Control*, 52(12):2254–2269, Dec.

[23] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Trans. Networking*, 3(2):226–244, Jun. 1995.

[24] N. U. Prabhu. *Foundations of Queueing Theory*. Kluwer Academic Publishers, 1997.

[25] L. Qian, Y.-J. Zhang, and M. Chiang. Distributed nonconvex power control using gibbs sampling. *IEEE Trans. Commun.*, 60(12):3886–3898, Dec. 2012.

[26] L. Rao, X. Liu, L. Xie, and W. Liu. Reducing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment. In *IEEE Infocom*, 2010.

[27] S. Ren, Y. He, and F. Xu. Provably-efficient job scheduling for energy and fairness in geographically distributed data centers. In *ICDCS*, 2012.

[28] C. Robert and G. Casella. *Monte Carlo Statistical Methods*. New York: Springer-Verlag, 2004.

[29] Y. Song, C. Zhang, and Y. Fang. A game theoretical analysis of joint channel and power allocation in wireless mesh networks. *IEEE J. Sel. Areas Commun.*, 26(7):1149–1159, Sep. 2008.

[30] Q. Tang, S. K. S. Gupta, and G. Varsamopoulos. Thermal-aware task scheduling for data centers through minimizing heat recirculation. In *CLUSTER*, 2007.

[31] H. P. Young. *Individual Strategy and Social Structure*. Princeton University Press, 1998.

[32] 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.