Sustainable Computing: Informatics and Systems 6 (2015) 3–16

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Distributed temperature-aware resource management in virtualized data center

Mohammad A. Islam[a,*], Shaolei Ren[a], Niki Pissinou[a], A. Hasan Mahmud[a], Athanasios V. Vasilakos[b]

[a] Florida International University, Miami, FL 33199, United States
[b] National Technical University of Athens, Athens 106 80, Greece

ARTICLE INFO

Article history:
Received 16 November 2013
Accepted 14 March 2014

Keywords:
Capacity provisioning
Data center
Distributed optimization
Load distribution
Temperature constraint

ABSTRACT

While resource consolidation enables energy efficiency in virtualized data centers, it results in increased power density and causes excessive heat generation. To prevent servers from overheating and avoid potential damage and/or service outages, data centers need to incorporate temperature awareness in resource provisioning decisions. Moreover, data centers are 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), virtual machine (VM) CPU allocation and load distribution for minimizing the data center power consumption while satisfying the Quality of Service (QoS), IT peak power and maximum server temperature constraints. By using DREAM, each server can autonomously adjust its *discrete* processing speed (and hence, power consumption, too), and optimally decide the VM CPU allocation as well as amount of workloads to process in the hosted VMs, in order to minimize the total power consumption which incorporates both server power and cooling power. We formally prove that DREAM can yield the minimum power with an *arbitrarily* high probability while satisfying the peak power and server temperature constraints. To complement the analysis, we perform a simulation study and show that DREAM can significantly reduce the power consumption compared to the optimal temperature-unaware algorithm (by up to 33%) and equal load distribution (by up to 86%).

1. Introduction

The rapid expansion of Internet-based services, together with the demand for scalable and robust infrastructures, has resulted in thriving number and size of virtualized data centers in recent years. While virtualization offers improved "power proportionality" through resource consolidation, the increasing number of virtualized data centers still accounts for a huge electricity consumption, and raises serious economic and environmental concerns due to the accompanying high electricity bill and carbon footprint. Hence, data center operators are constantly urged to reduce the power consumption while maintaining a premium Quality of Service (QoS).

For a data center serving delay-sensitive workloads such as web services, lowering energy consumption without affecting the QoS is challenging. Although data center operation has attracted significant interest from the research community and undergone a substantial improvement [1], there still exist some hurdles that limit the data center operations. First, data centers have a stringent IT peak power budget: exceeding the peak power constraint will result in serious consequences such as service outages and equipment damages [2–4]. Increasing the power budget, however, may not be an immediate or viable solution in practice, as the peak power budget is often determined during the construction phase and 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) [4]. 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

* Corresponding author. Tel.: +1 3053482032.
E-mail addresses: misla012@fiu.edu (M.A. Islam), sren@fiu.edu (S. Ren), pissinou@fiu.edu (N. Pissinou), amahm008@fiu.edu (A.H. Mahmud), vasilako@ath.forthnet.gr (A.V. Vasilakos).

http://dx.doi.org/10.1016/j.suscom.2014.03.002
2210-5379/

overheating that could potentially induce server damages and huge economic losses [5]. 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), virtual machine (VM) CPU allocation and load distribution for minimizing the data center power consumption while satisfying the QoS, 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* [5,6]), DREAM makes distributed decisions while proactively taking into account the potential impact of the decisions on the inlet temperature increase to avoid server overheating. While total power is minimized for the data center operation by incorporating the server power and cooling system power in the optimization objective, the QoS constraint, quantified by average delay threshold, guarantees the overall service quality. By using DREAM, each server can autonomously adjust its *discrete* processing speed (and hence, power consumption, too) and optimally decide the VM CPU allocation and amount of workloads to process, in order to minimize the total power consumption. DREAM builds upon a variation of Gibbs sampling technique [7], combined with dual decomposition [8]. We conduct a rigorous performance analysis and formally prove that DREAM can yield the minimum power, while satisfying the QoS, peak power and server 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 power by up to 33%.

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, CPU allocation for the hosted VMs and the amount of workloads to process, while maintaining QoS and incorporating three important practical constraints: discrete processing speeds, peak server power, and maximum server inlet temperature constraints. It is rigorously proved that DREAM minimizes the total power with an arbitrarily high probability.
2. We conduct a comprehensive simulation, and the results show that DREAM achieves a significantly lower power consumption compared to two widely used existing algorithms.

The rest of this paper is organized as follows. The model is described in Section 2. In Section 3 and 4, we present the problem formulation and develop our distributed online algorithm, DREAM, respectively. Section 5 provides a simulation study to validate DREAM. Related work is reviewed in Section 6, and finally, concluding remarks are offered in Section 7.

2. Model

We consider a model in which the capacity provisioning, VM CPU allocation and workload distribution decisions are updated periodically and the period is sufficiently large (e.g., 1 h) such that the room temperature can become stabilized. Moreover, at the beginning of each decision period, the data center operator can predict the workload arrival rate over the period. For example, each decision period corresponds to 1 h if the data center leverages hour-ahead workload arrival prediction that is readily available in practice [4,9,10]. 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 1.

Table 1
List of notations.

Notation	Description
x_i	Speed of server i
$c_{i,j}$	CPU allocation for $VM_{i,j}$
$\mu_{i,j}$	Service rate of $VM_{i,j}$
$\lambda_{i,j}$	Workloads distributed to $VM_{i,j}$
p_i	Average power consumption of server i
p	Total power consumption
$d_{i,j}$	Average delay in $VM_{i,j}$
T_{sup}	Supply temperature
T_{out}	Outside air temperature
$\mathbf{T}_{\text{in}}$	Server inlet temperature
$\mathbf{D}$	Heat transfer matrix
N	Number of servers
M	Number of server racks
J	Number of job types

2.1. Data center

We consider a data center that has N physical servers that are mounted in M racks (or chassis). Each server hosts multiple VMs to serve different types of workloads. Without causing ambiguity, we also use *servers* to represent physical servers wherever applicable. The mth 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 mth rack by $\mathcal{N} = \{1, 2, \ldots, N\}$ and $\mathcal{N}_m$, respectively, and it follows naturally that $\mathcal{N} = \mathcal{N}_1 \bigcup \mathcal{N}_2 \bigcup \ldots \bigcup \mathcal{N}_M$ 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 [11]), or switching servers on/off. 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., power supply system) which, however, can be conveniently absorbed by a (partial) power usage effectiveness (PUE) factor [9].

2.1.1. Server power

As computing takes up a large portion (typically 40%) of server power consumption [3], adjusting CPU speed can significantly affect the total power consumption. Hence, we focus on CPU resource allocation, while treating other resources (e.g., memory, disk) as sufficient and *non-bottleneck* resources. Although this assumption may not hold for all application scenarios (e.g., memory/disk power consumption may vary considerably for I/O-intensive workloads), we note that it is reasonably accurate for CPU-intensive workloads that are the main concentration of our study [12,13].

To keep our model general, we consider that server i can choose its speed x_i out of a finite set $S_i = \{s_{i,0}, s_{i,1}, \ldots, s_{i,L_i}\}$, where $s_{i,0} = 0$ represents zero speed (server deep sleep or shut down) and L_i is the number of available positive speed settings. The speed x_i quantifies

[1] Server inlet temperature is the temperature of air entering/exiting 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).

server i's total CPU resource (measured in, e.g., GHz) that is available for processing VM workloads (besides the CPU consumption by the privileged domain0/root VM handling resource management). Considering that the servers consume no power when shut down (i.e., under the zero-speed mode), we express the average power consumption and peak power of server i, respectively, as [12].

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

$$\hat{p}_i(x_i) = p_{i,s} + p_{i,c}(x_i), \tag{2}$$

where u_i is CPU utilization of server i (that will be specified in the next section), $p_{i,s}$ 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 operating at a speed of x_i, and the indicator function $\mathbf{1}_{(x_i>0)} = 1$ if and only if the processing speed $x_i > 0$.

2.1.2. Cooling power

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 [14]. Modern data centers often leverages multiple cooling approaches and, like in [15], we consider a cooling system consisting of two cooling approaches: *outside air cooling* and water-cooled *computer room air conditioner* (CRAC). In what follows, we describe the two cooling approaches and formulate their power consumption.

In outside air cooling, cold outside air is directed into the data center using air-side economizer to cool down servers and the hot exhaust air is rejected out without recirculation. While outside air cooling is economic and can save significant cooling energy [16], its energy efficiency heavily depends on the outside air temperature denoted by T_{out} [15,17]. We consider that for a target server room supply temperature of T_{sup} (which is also the "cold isle" temperature), the data center uses outside air cooling when $T_{\text{out}} \le T_{\text{sup}}$ and CRAC cooling when $T_{\text{out}} > T_{\text{sup}}$.[2] The outside air cooling power is mainly the power consumption of the blower fans and generally can be expressed as a cubic function of blower speed, which is proportional to the cooling load (i.e. IT power) given a fixed outside temperature and target supply temperature [15,18]. However, a more power efficient design is to use large blowers running at low speed instead of small blowers running at maximum speed, for which the outside air cooling power can be approximated as an affine function of the IT power [19].

$$p_{air} = k(\delta T) \times \sum_{i=1}^{N} p_i(u_i, x_i), \tag{3}$$

where $k(\delta T)$ is the proportionality constant that depends on the difference between outside air and target supply temperature, denoted by $\delta T = T_{\text{sup}} - T_{\text{out}}$ [15], and $\sum_{i=1}^{N} p_i(u_i, x_i)$ is the total power of all the servers in the data center. Intuitively, $k(\delta T)$ decreases with increase in δT and vice versa. This is because for colder outside air, less air needs to be pumped inside the data center to cool down the servers. To characterize the efficiency of the cooling system, we define *coefficient of performance* (CoP) [20], which is the ratio of heat removed (IT power) to the work necessary to remove the heat (cooling power). From (3), we can write CoP for outside air cooling as

$$\text{CoP}_{\text{air}}(T_{\text{out}}, T_{\text{sup}}) = \frac{\sum_{i=1}^{N} p_i(u_i, x_i)}{p_{air}} = \frac{1}{k(\delta T)} \tag{4}$$

On the other hand, for water-cooled CRAC, the CoP is almost independent of ambient temperature T_{out} and depends only on T_{sup} [20]. While each CRAC might have its own unique power efficiency model, as an example, we use the CoP model of HP Labs Utility Data Center presented in [20].

$$\text{CoP}_{\text{CRAC}}(T_{\text{sup}}) = 0.0068T_{\text{sup}}^2 + 0.0008T_{\text{sup}} + 0.458. \tag{5}$$

The effective CoP of the data center considering both outside air and CRAC cooling can therefore be modeled as

$$\text{CoP}(T_{\text{out}}, T_{\text{sup}}) = \begin{cases} \text{CoP}_{\text{air}}, & \text{when } T_{\text{out}} \le T_{\text{sup}}, \\ \text{CoP}_{\text{CRAC}}, & \text{otherwise.} \end{cases} \tag{6}$$

While we use the above CoP model for our data center as an example, more complex models involving optimum operation choice and/or combination of outside air and CRAC cooling [15] can also be incorporated without affecting our approach of solution.

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[3] is given by

$$p(\vec{u}, \vec{\mathbf{x}}) = \sum_{i=1}^{N} p_i(u_i, x_i) \cdot \left[1 + \frac{1}{\text{CoP}(T_{\text{out}}, T_{\text{sup}})}\right], \tag{7}$$

where $\vec{u} = (u_1, u_2, \ldots, u_N)$ and $\vec{\mathbf{x}} = (x_1, x_2, \ldots, x_N)$ are the server utilization and capacity provisioning decisions, respectively. Our study can easily capture the electricity cost by multiplying (7) with the electricity price. Note that, since the decisions are updated infrequently (e.g., hourly) as in [9,21], 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 [10].

2.2. Virtual machine

Virtualization provides abstraction of the underlying hardware to upper layers by creating a set of virtualized system platforms on which a customized operating system can run. In a virtualized environment, each physical server hosts a set of VMs and contains a VM monitor (VMM, also called hypervisor) that is responsible for allocating hardware resources to the hosted VMs. In our study, we consider the data center serves J types of workloads which are processed by VMs placed in different servers. Each server type can host up to J number of VMs and each VM is assigned a fraction of the total available CPU resource of that server. We denote this VM serving type-j workload at server i by VM_{ij} and its CPU resource allocation by $0 \le c_{ij} \le x_i$.

2.3. Workload

We focus on delay-sensitive workloads/jobs (as in [9,12]), whereas delay-tolerant batch workloads can be easily captured by maintaining a separate batch job queue as considered by several existing studies [12]. There are J types of workloads, and we denote by $a_j \in [0, a_{j\max}]$ the arrival rate of type-j workloads during time t. As assumed in prior work [9,10,22], the value of a_j is accurately available at the beginning of each time slot t, as can be achieved by using various techniques such as regression analysis for workloads that exhibit daily/weekly patterns, while the robustness against inaccurate knowledge of the service rates/workload arrival rates is demonstrated in the simulation.

Each type of workload is distributed among the VMs placed in different servers. Let λ_{ij} be the j type workload processed at

[2] A supply temperature higher than the outside temperature can be easily achieved by mixing the outside air with hot exhaust air.

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

server i. Thus, the workload of $VM_{i,j}$ is $\lambda_{i,j}$. The service rate (i.e., how many jobs can be processed in a unit time) of $VM_{i,j}$ is given by $\mu_{i,j} = \mu_{i,j}(c_{i,j},)$. The function $\mu_{i,j}(\cdot)$ maps the allocated CPU resource to the service rate. While in general it is non-trivial to accurately obtain the mapping $\mu_{i,j}(\cdot)$ [23,24], we note that the service rate of CPU-intensive jobs can be approximated as an *affine* function of the allocated CPU resource (provided that memory, disk, etc. are non-bottleneck) [13]. Thus, as in [12], we assume in the following analysis that $\mu_{i,j}(\cdot)$ is exogenously determined, for $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, J$, while noting that modeling the function $\mu_{i,j}(\cdot)$ can be done using the techniques developed in [23,24] that are beyond the scope of our study.

To quantize the *overall* delay performance, we use average delay at $VM_{i,j}$ represented by a convex function $d_{i,j}(\lambda_{i,j}, \mu_{i,j})$, which is intuitively increasing in $\lambda_{i,j}$ and decreasing in $\mu_{i,j}$ [9,10]. As a concrete example, we model the service process at each VM as an M/G/1/PS queue and it is well known [25] that the average delay of such systems can be written as[4]

$$d_{i,j}(\lambda_{i,j}, \mu_{i,j}) = \frac{1}{\mu_{i,j} - \lambda_{i,j}}, \tag{8}$$

Here, we ignore the network delay cost that can be approximately modeled as a certain constant [9] and added into (8) without affecting our approach of analysis. It should be made clear that although the M/G/1/PS queuing model may not capture the exact response time, it is widely used as an analytic vehicle to guide resource management decisions [4,9]. Essentially, setting a constraint on (8) will limit the utilization of each VM, thereby translating into a certain performance constraint [23]. Finally, we note that the delay model in (8) implicitly assumes that VMs are *perfectly* isolated without interfering with each other. While this may not be true in heavy traffic regimes (e.g., due to cache, I/O contention) as pointed out the existing research [23], we consider perfect isolation and use the delay model only as an *approximate* indication for the actual delay performance (as studied in [12]). In other words, the model is only intended to facilitate the design of our algorithm, while the actual decision and delay performance should still be appropriately calibrated online to account for the factors (e.g., VM interference, imperfect workload monitoring) that are not captured by our model.

3. 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.

3.1. Objective and constraints

At the beginning of each decision period, the data center updates the server CPU speed, VM CPU allocation and workload distribution to minimize the operational 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). As electricity cost is a major component of operational cost and depends on the power consumption, we use the total power consumption as a proxy of operational cost, and hence our optimization objective function is given by (7). In the rest of the paper, we use "cost" and "power" interchangeably to represent the operational cost of the data center. Note that if time-varying electricity price is utilized (e.g., via smart grid), our approach still applies by multiplying (7) with the electricity price.

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

$$x_i \in \mathcal{S}_i = \{s_{i,0}, s_{i,1}, \ldots, s_{i,L_i}\}, \quad \forall i. \tag{9}$$

The power distribution system (consisting of the power supply/distribution units, failure backup generator, etc.) of a data center is typically subject to a *predetermined* maximum operating power capacity that cannot be easily increased without infrastructure expansion. In addition, data centers are often partly billed based on their peak power usage, although the peak usage charge is not explicitly taken into consideration in our study. Thus, in light of that exceeding the peak power limit results in fines and/or increased financial charges, we capture in our formulation the peak power constraint as follows:

$$\sum_{i=1}^{N} \hat{p}_i(x_i) \le \hat{P}, \tag{10}$$

where $\hat{P}$ is the peak (server) power constraint.

As delay performance is very important for a data center, which affects customer satisfaction and revenue, we add a threshold on the average delay for the VMs.

$$d_{i,j} \le d_{th}, \forall i, \forall j \tag{11}$$

where d_{th} is the maximum allowed average delay for any VM. The VM resource allocation needs to satisfy

$$\sum_{j=1}^{J} c_{i,j} \le x_i, \forall i. \tag{12}$$

To avoid workload dropping the load distribution must satisfy

$$\sum_{i=1}^{N} \lambda_{i,j} = a_j, \forall j \tag{13}$$

Now, we are ready to derive the average server utilization u_i as follows:

$$u_i = \sum_{j=1}^{J} \frac{\lambda_{i,j}}{\mu_{i,j}} \cdot \frac{c_{i,j}}{x_i} \tag{14}$$

where $\lambda_{i,j}/\mu_{i,j}$ is the utilization of $VM_{i,j}$ and $c_{i,j}/x_i$ is portion of server i's CPU resources allocated to $VM_{i,j}$. This completes the expression of average server power consumption[5] in (1).

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., in the order of minutes [14]) 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 [6,14,26] has shown that the relation between the average server power consumption and hot duct inlet temperature is governed by the following equation:

$$\mathbf{T}_{\text{in}} = \mathbf{T}_{\text{sup}} + \mathbf{D} \times \mathbf{p}, \tag{15}$$

[4] The 95%-percentile delay of an M/M/1 queue is also $\frac{1}{\mu_{i,j}-\lambda_{i,j}}$ [10].

[5] The domain0/root VM power consumption is absorbed in the "static" power $p_{s,i}$ of server i.

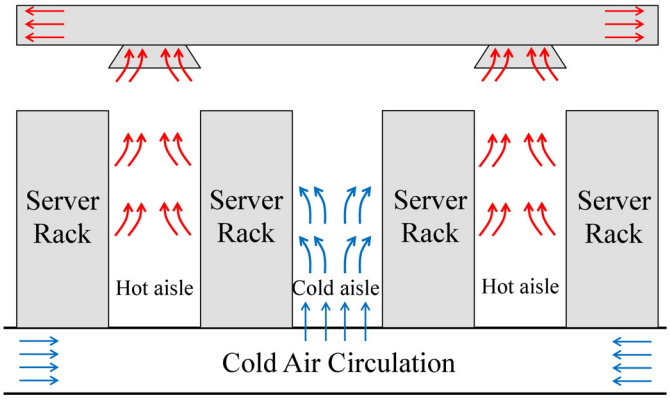


Fig. 1. Heat recirculation inside server room.

where $\mathbf{T}_{\text{in}} = (T_{1,\text{in}}, T_{2,\text{in}}, \ldots, T'_{M,\text{in}})$ and $\mathbf{T}_{\text{sup}} = (T_{\text{sup}}, T_{\text{sup}}, \ldots, T_{\text{sup}})'$ 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 $\mathbf{p} = (\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_M)'$ in which $\overline{p}_m = \sum_{i \in \mathcal{N}_m} p_i(u_i, x_i)$ is the total average power consumption of servers mounted in the mth 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 ith rack caused by the servers in the jth rack due to heat recirculation. While in some state-of-the-art data centers [27], *heat containment* techniques are deployed to reduce heat recirculation by isolating the hot and cold air in the aisles (by covering either the hot aisles or cold aisles) or even inside the racks, the heat transfer matrix model is still applicable but the non-diagonal matrix elements have much smaller values: for an ideal case with no heat recirculation, the $\mathbf{D}$ becomes a diagonal matrix.

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:

$$\mathbf{T}_{\text{in}} \preceq \mathrm{T}, \tag{16}$$

where $\mathrm{T} = \{\hat{T}_1, \hat{T}_2, \ldots, \hat{T}_M\}$ 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., $\hat{T}_1 = \hat{T}_2 = \cdots = \hat{T}_M = \hat{T}$.

3.2. Optimization problem

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

$$\mathbf{P1}: \min_{\mathcal{A}} p(\vec{\mathbf{x}}, \vec{c}, \vec{\lambda}) \tag{17}$$

$$s.t., \text{constraints (9)–(13), (16)}, \tag{18}$$

where $\mathcal{A}$ represents all the feasible server speed, VM CPU allocation and load distribution decisions that we need to optimize. As server utilization is a function of VM CPU allocation and workload distribution, the power consumption in (17) is presented as a function of $\vec{c}$ and $\vec{\lambda}$. While incorporating real-time feedback from the environment (e.g., temperature monitoring result [5]) 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 (18). Note that, although our main focus is on optimizing server speed, VM CPU allocation 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 (7). 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 (9), **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) [8,28]. 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 4, in which each server autonomously makes capacity provisioning and load distribution decisions.

4. 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.

4.1. DREAM

Algorithm 1. DREAM

1: Initialization: each server i chooses a feasible processing speed x_i^*, VM CPU allocation $c_{i,j}^*$ and load distribution $\lambda_{i,j}^*$, for $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots J$; set $\vec{\mathbf{x}} \leftarrow \vec{\mathbf{x}}^*$, $\vec{c} \leftarrow \vec{c}^*$, $\vec{\lambda} \leftarrow \vec{\lambda}^*$, $\tilde{p} \leftarrow \infty$
2: Obtain $\vec{\lambda}^*$ and $\vec{c}^*$ by solving

$$\min_{\vec{\lambda}, \vec{c}} p(\vec{\mathbf{x}}^*, \vec{c}, \vec{\lambda}), \tag{19}$$

subject to (11),(12),(13), (16), and set $\tilde{p}^*$ to the minimum value of (19); set $\tilde{p}^* \leftarrow \infty$ if minimizing (19) subject to (11), (12),(13), (16) is not feasible
3: $\omega \leftarrow \dfrac{\exp\left(\frac{\delta}{\tilde{p}^*}\right)}{\exp\left(\frac{\delta}{\tilde{p}}\right) + \exp\left(\frac{\delta}{\tilde{p}^*}\right)}$
4: With a probability of ω: each server i sets $x_i \leftarrow x_i^*$, $c_{i,j} \leftarrow c_{i,j}^*$, $\lambda_{i,j} \leftarrow \lambda_{i,j}^*$ and $\tilde{p} \leftarrow \tilde{p}^*$; with a probability of $1 - \omega$: each server i sets $x_i^* \leftarrow x_i$
5: Randomly select a server i; server i randomly selects a processing speed $x_i' \in \mathcal{S}_i$ and sets $x_i^* \leftarrow x_i'$
6: Return $\vec{\mathbf{x}}$, $\vec{c}$ and $\vec{\lambda}$ if the stopping criterion is satisfied; otherwise, go to Line 2

As aforementioned, **P1** is an optimization problem involving mixed-integer nonlinear programming. While there exist various centralized techniques (such as Generalized Benders Decomposition [29]) 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 [7,30], 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 VM CPU allocation and 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 2 in DREAM, i.e., minimizing (19), can be solved distributedly using dual decomposition[8]. 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 5, 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 [7,28]. To avoid the inefficiency, DREAM explores new solutions by *deliberately* introducing randomness to decision making (i.e., line 4), even though they may be worse than the current solution [30]. 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., $\tilde{p}^* \leq \tilde{p}$). 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 4 is a variation of Gibbs sampling [7,30]: in line 4, 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.

4.2. Analysis

In this section, 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 virtualized data centers [31]). We will illustrate in the next section the iteration process of DREAM under different values of δ.

Performance. Theorem 1, whose proof outline is provided in A, 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 (17) subject to (11),(12),(13),(16). When $\delta \to \infty$, Algorithm 1 converges to the globally optimal solution with a probability of 1.*

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).[6] 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 cooling system 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 [6,14,26]. Now, we extend DREAM to incorporate the supply temperature T_{sup} as an additional decision variable. Specifically, we denote by $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$ the (discrete) set of available supply temperatures that are available to the cooling system.[7] 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.

5. Simulation

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

5.1. Settings

As in prior work (e.g., [6,14,26]), 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 (i.e. server speed and VM CPU allocation) 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 a cooling system (which can switch to air economizer or water-cooled CRAC, depending on the desired supply temperature and outside air temperature) and five racks installed in a row, each consisting of 48 servers. Note that a large data center can consist of multiple smaller server rooms, each of which may have its own cooling 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 [33] and listed in Table 2. The total peak server power constraint is 65 KW.

[6] A more rigorous analysis of convergence rate for the original Gibbs sampling technique in the context of wireless networks is available in [32].

[7] Continuous supply temperatures can be treated in a similar way [32].

Table 2
Power consumption and normalized speed.

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

Virtual machines: We consider that the servers can host VMs to serve any type of workloads. The VM service rate (i.e., maximum number of jobs served per unit time) is proportional to the allocated CPU resource, while the proportionality factor depends on the workload type as well as the server type: in our simulation, we use 8 different constants of proportionality to simulate 4 different types of jobs served by 2 different types of servers in the data center. In particular, the constants of proportionality are pre-determined and chosen from the following set: $48 * [0.90, 0.95, 1.00, 1.05, 1.10, 1.15, 1.20, 1.25]$. For example, if a workload has a constant of proportionality of 48.00 and is processed on a VM with a normalized CPU resource of 1.0, then the VM can process 48 "jobs" of this workload on average per time slot (i.e., hour in our study).

Workloads: We consider that the data center serves 4 different types of jobs. The workloads for our simulations are taken from the research publication [34] and originally profiled from I/O traces taken from 6 RAID volumes at Microsoft Research (MSR) Cambridge. The traced period was 1 week starting from 5 PM (GMT) on February 22, 2007 [34]. We add 50% random variation to the one week trace to generate the workload of the 4 types of jobs. Fig. 2(a) illustrates the workload for type-1 jobs, where the workloads are normalized with respect to the total maximum service capacity of the data center. We also 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, and use the trace in our sensitivity study.

Others: Following [14,26,35], we set the 5×5 heat transfer matrix **D** as follows: $D_{i,j} = \frac{0.7^{|i-j|}}{3500}$, for $i, j = 1, 2, \ldots, 5$. By default, the supply temperature and maximum inlet temperature constraints are 20 °C (unless optimized) and 30 °C, respectively. The unit of power consumption is Watt. The default smoothing parameter δ in Algorithm 1 is 1500, which gives us a good performance at a reasonable convergence rate (see Fig. 3 for more details). By default, the normalized average delay constraint for all workloads is 1.0, as obtained by plugging in the normalized service/arrival rates into the delay model (8). The outside temperature data used in the simulation is the hourly air temperature of first week of June, 2013 recorded in Tokyo, Japan [36,37] and shown in Fig. 2(b).

5.2. Simulation results

In this subsection, we first show the iteration process of executing DREAM with different starting points and smoothing factors. Then we introduce two benchmark algorithms and show some insight into the resource management decisions made by DREAM in different workload intensities. Next, we compare performance of DREAM with the two benchmark algorithms. We then study the impact of different parameters on these algorithms and finally do a sensitivity study with different workload and prediction error in the heat transfer matrix **D**.

5.2.1. Execution of DREAM

We show the iteration process of executing DREAM in Fig. 3 under default settings and with a moderate arrival rate equal to 50% of maximum capacity. For illustration purposes, all the servers within one rack are assumed to choose the same speed (i.e., one server chooses the service rate on behalf of the whole rack) and VM CPU allocation. Fig. 3(a) and (b) show the iteration of the cost $p(\vec{\mathbf{x}}, \vec{c}, \vec{\lambda})$ 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 #2,3,4 are shown in Fig. 3(b). Note that for the given workload, servers in rack #3 are shut down even though they have the same configuration as those in rack #1,2, because they are mounted on the rack placed in the middle and will result in a more significant temperature increase due to heat recirculation. We also observe that during the iterations, DREAM explores bad solutions that even *temporarily* increases the cost. Nonetheless, such exploration is essential to avoid being completely trapped in a locally optimal solution [28]. 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.

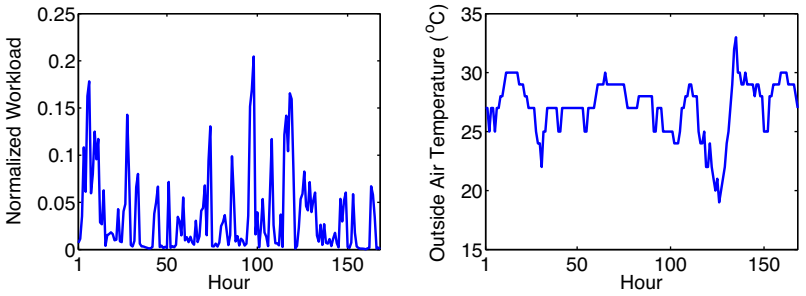
5.2.2. Comparison with existing algorithms

We compare DREAM with two existing algorithms specified as follows.[8]

EQL (Equal load distribution): CPU allocation to VMs are made proportional to the workload arrival rates. Workloads are equally distributed across all the VMs in the active servers. Server speeds are selected in a greedy manner subject to the peak power constraint (i.e., the speeds 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 and does not consider supply temperature as a variable.

EQL is a variant of the temperature-reactive algorithm in [6] 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 20 °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 4. 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.

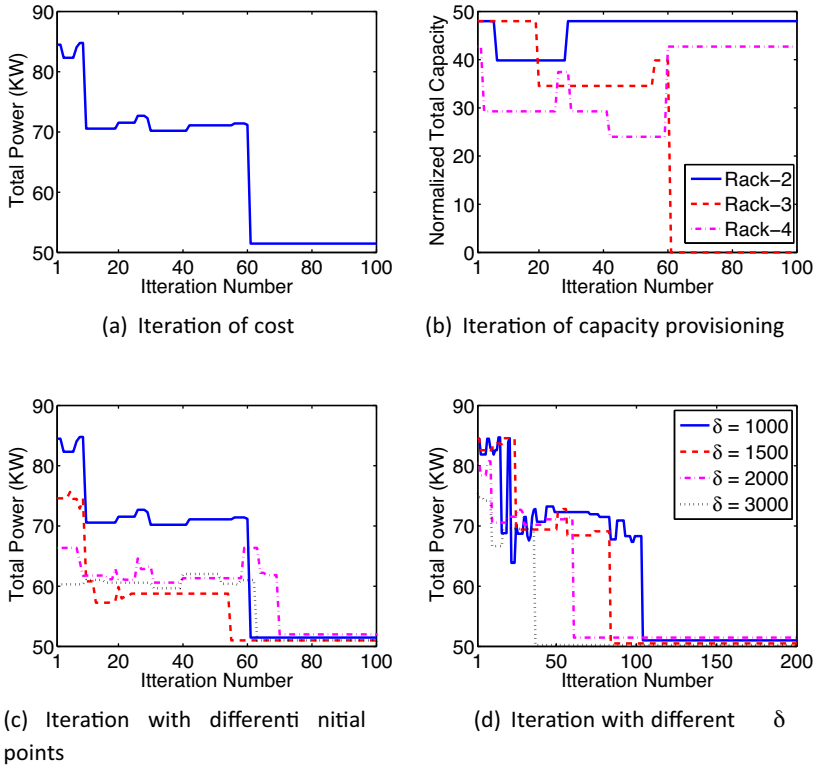
[8] While our heat recirculation model follows [26], the scheduling algorithm in [26] is intended for HPC jobs and does not apply to our considered delay-sensitive workloads.



(a) Workload ofj ob type-1

(b) Outside air temperature of Tokyo, Japan [36, 37].

Fig. 2. Data set.



(a) Iteration of cost

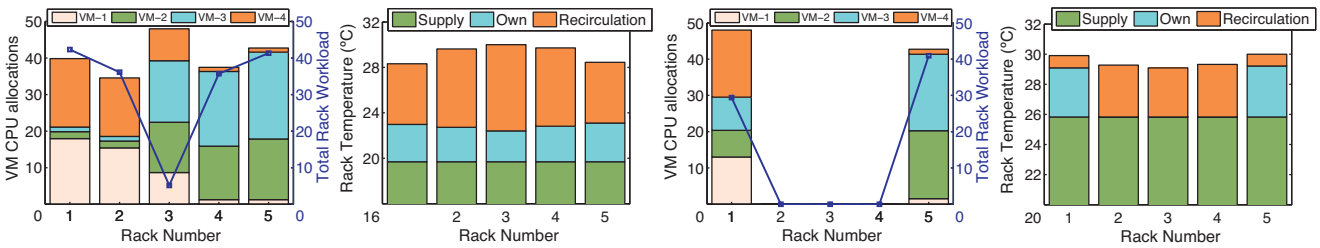
(b) Iteration of capacity provisioning

(c) Iteration with differenti nitial points

(d) Iteration with different δ

Fig. 3. Execution of DREAM.

Resource allocation. Now, we show the resource allocation and workload distribution decisions made by DREAM and TempU, and the resulting rack temperatures for different workload intensities in Figs. 4 and 5. Specifically, we use total workload equal to 70% of maximum total capacity as *high workload* and 30% of maximum total capacity as *low workload*. In the resource allocation figures, height of the stacked bar represents the normalized total CPU speed of all the servers in the rack, while the fractions of total CPU allocated to the VMs are identified by the stacks. We see in Fig. 4(a) and (b) that for a high workload, DREAM turns on all the servers but



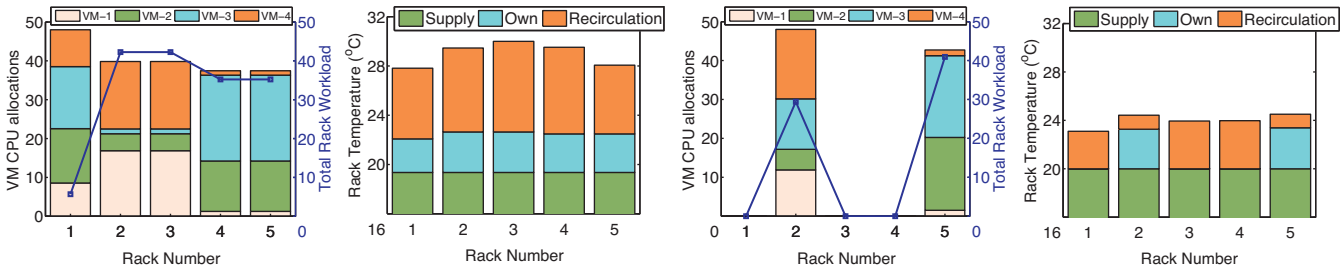
(a) Resource allocation - High workload

(b) Rack temperature - High workload

(c) Resource allocation - Low workload

(d) Rack temperature - Low workload

Fig. 4. DREAM.



(a) Resource allocation - High workload (b) Rack temperature - High workload (c) Resource allocation - Low workload (d) Rack temperature - Low workload

Fig. 5. TempU.

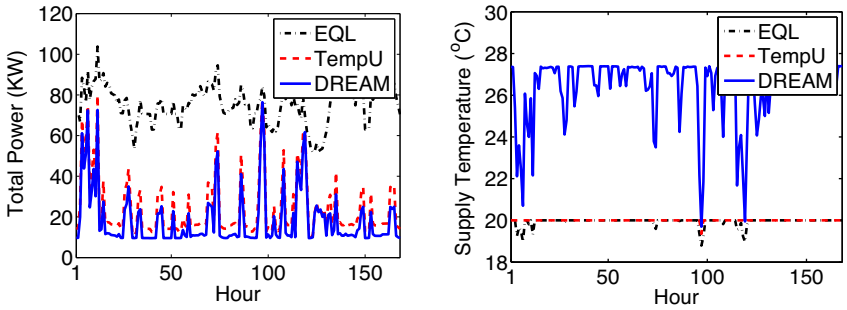


Fig. 6. Performance comparison.

processes the lowest workloads in the middle rack which has the highest recirculation temperature increase. On the other hand, the temperature-oblivious TempU sends high workloads to the middle racks, and has to reduce the supply temperature during runtime to make up for the temperature increase, as seen in Fig. 5(a) and (b). For low workload, DREAM turns on only two server racks and increases the supply temperature to reduce cooling power consumption, shown in Fig. 4(c) and (d). TempU in Fig. 5(c) and (d), also turns on two server racks but keep the supply temperature at default 20 °C, where the rack temperatures are well below the temperature constraint. These figures show how the temperature awareness in DREAM affects its resource provisioning decisions (i.e. server speed and VM CPU allocation) compared to temperature oblivious algorithm.

Hourly power reduction. We show in Fig. 6 hourly comparison between DREAM and the two existing algorithms using our one-week workload trace. Specifically, Fig. 6(a) demonstrates that DREAM can significantly reduce the total power consumption compared to TempU (by up to 33%) and EQL (by up to 86%). We also observe that, compared to EQL, the power 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. Next, we illustrate the hourly supply temperature in Fig. 6(b). It can be seen that, DREAM proactively increases the supply temperature from the default 20 °C during light workload to reduce cooling power consumption, whereas TempU and EQL keep the supply temperature at default value and only reduce it reactively when the workload is high.

5.2.3. *Impact of delay constraint*

In Fig. 7, we show the impact of average delay constraint on the performance of DREAM and TempU. The average delay in Fig. 7 is the *normalized* value based on the delay model (8), indicating the strictness of performance constraint. We run our simulation with a moderate workload (equal to 50% of total maximum capacity) for different average delay constraints. We see that the algorithms incur higher power consumption for lower delay constraint because, to reduce average delay the algorithms need to allocate

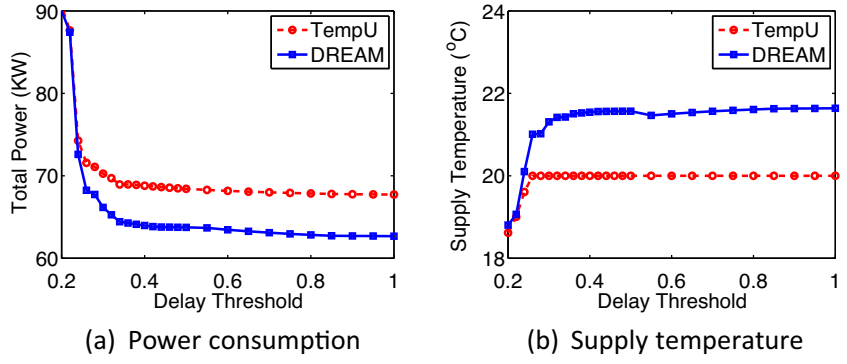


Fig. 7. Effect of delay constraint.

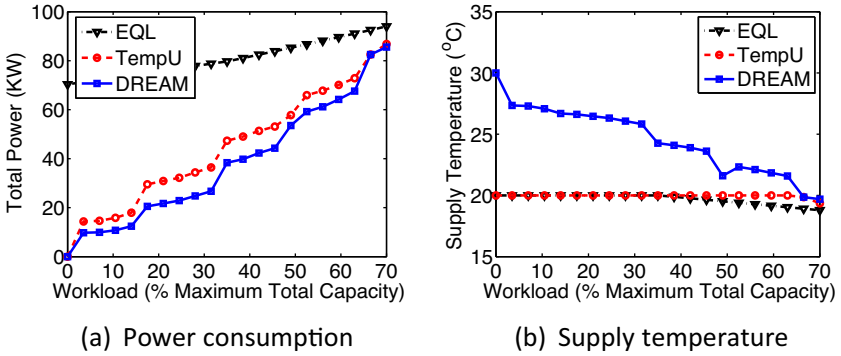


Fig. 8. Effect of workload.

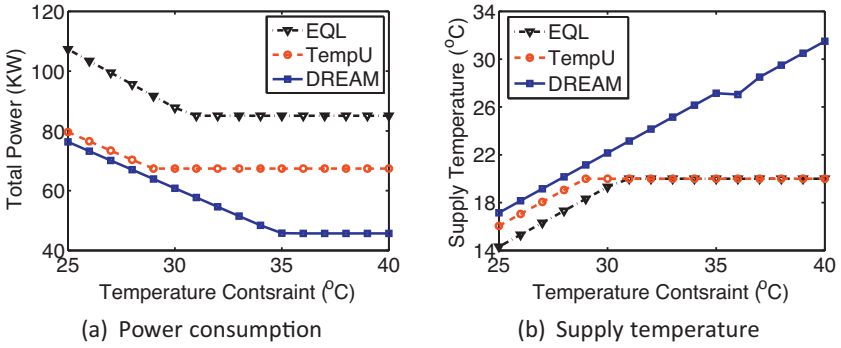


Fig. 9. Effect of temperature constraint.

more CPU resources to the VMs, thereby running the servers at higher speeds and consuming more power. In Fig. 7(a), we observe that except for very low delay constraint, DREAM has lower power than TempU. It is because, DREAM takes advantage of the relaxed delay constraint, and consumes less server power by reducing server speed as well as less cooling power by increasing supply temperature. On the other hand, TempU does not increase the supply temperature, as seen in Fig. 7(b), and incurs a higher cooling power cost.

5.2.4. *Impact of workload*

We vary the workload arrival rate to show its effect on the power consumption in Fig. 8 under the default maximum inlet temperature constraint of 30 °C. In Fig. 8(a), it can be seen that DREAM achieves a lower cost than TempU under all the workloads, TempU does not increase supply temperature during low workloads (which incurs higher cooling power), whereas DREAM can proactively schedule workloads to avoid server overheating while maintaining a relatively high supply temperature (as corroborated by Fig. 8(b)). EQL incurs the highest cost, because it only reactively shuts down overheated servers and thus incurs a large idle power consumption.

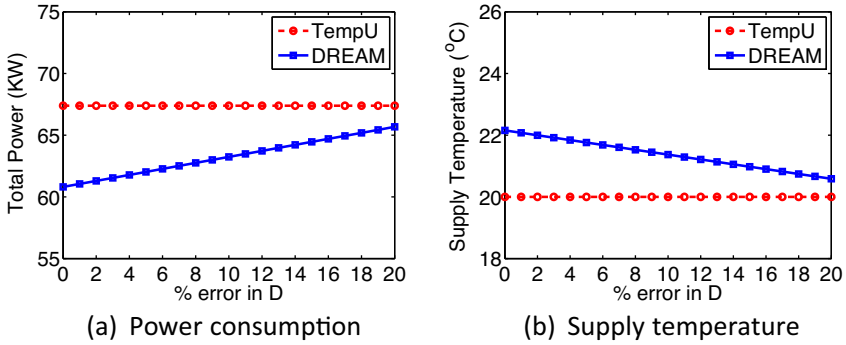
5.2.5. *Sensitivity study*

In this subsection, we present the results of sensitivity studies by varying the temperature constraint, incorporating errors in the heat transfer matrix **D**, using different workload pattern and workload overestimation.

Effect of temperature constraint. To assess the robustness of DREAM in terms of the temperature constraint, we show in Fig. 9 the effect of the maximum inlet temperature constraint[9] on the total power under a moderate workload of 50% of total capacity. Fig. 9(a) illustrates that as the temperature constraint increases, the power consumption 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 29 °C, because the default temperature of 20 °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, which is shown in Fig. 9(b). Similar behavior is also observed for EQL, for which power reduction saturates for temperature constraint at approximately 32 °C. Even though the supply temperature is optimized for TempU (e.g., when the temperature constraint is lower than 29 °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 (for server and VM) and load distribution decisions on the inlet temperature increase.

Errors in D. The heat transfer matrix **D** is crucial for DREAM to make optimal decisions to avoid server overheating. In practice, **D** is derived by measurement combined with CFD analysis [14,26,35], and thus, it may not be *accurately* obtained. To address the errors in **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. 10(a) shows the result under a moderate workload; DREAM achieves a lower cost than TempU even though there are 20% errors

[9] 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).



(a) Power consumption (b) Supply temperature

Fig. 10. Effect of error in D.

in obtaining the heat transfer matrix **D**. We also see the effect of error in **D** in the supply temperature in Fig. 10(b), where DREAM chooses a lower supply temperature as the error increases. This is because for the conservative approach of increased values in **D**, DREAM predicts a higher inlet temperature due to the increased heat recirculation, and thus acts to satisfy the temperature constraint by lowering the supply temperature. TempU ignores the temperature constraint during its optimization and hence achieves a constant power and supply temperature regardless of the errors in **D**.

Different workload trace. We conduct our simulation with FIU workload trace which shows a different workload pattern than MSR workload. We do an hourly comparison of DREAM with TempU and EQL similar to what we show in previous subsection. In Fig. 11(a), we see that DREAM have a lower power consumption than the other two algorithms: up to 29% lower than TempU and up to 71% lower than EQL. The supply temperature curves in Fig. 11(b) shows that DREAM proactively increase the supply temperature during low workload to reduce cooling power. These figures demonstrate DREAM can effectively reduce power consumption regardless of workload type.

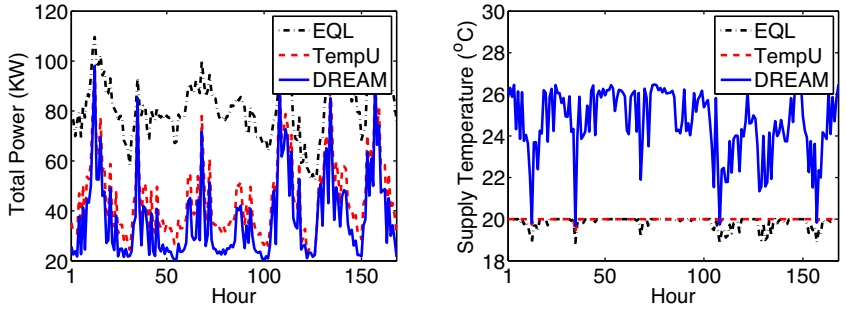
Workload overestimation. Since in practice it may not always be possible to accurately predict the hour-ahead workload arrivals, a conservative approach to coping with unexpected traffic spikes can be overestimating the workload arrival rate and turning on more servers. In essence, workload overestimation is equivalent to imperfect modeling of VM service rate. In Fig. 12, we show the increases in the total power caused by workload overestimation for both high and moderate workloads, respectively. Naturally, workload overestimation results in capacity over-provisioning as well as decreased supply temperature in the anticipation of potentially higher amount of heat generated by servers. Both of these factors contribute towards the increase of total power. We see from Fig. 12 that even with workload overestimation of 10%, DREAM still outperforms TempU consistently in terms of the total power consumption under both high and moderate workloads, demonstrating that DREAM is robust against workload prediction errors.

6. Related work

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

Data center optimization and VM resource allocation. There has been a growing interest in optimizing data center operation from various perspectives such as cutting electricity bills [9,21,25,38] and minimizing response times [4,10]. 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 reducing the energy cost of data centers [38]. As data centers are becoming increasingly virtualized, VM resource management has attracted much research interest: e.g., [39] studies energy-efficient load balancing for web services; [12] proposes admission control and dynamic CPU resource allocation to minimize the cost while bounding the queueing delay for batch jobs; [40–42] study various dynamic VM placement and migration algorithms that may be combined with our proposed solution. These studies assume server CPU 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



(a) Power consumption (b) Supply temperature

Fig. 11. Performance with different workloads.

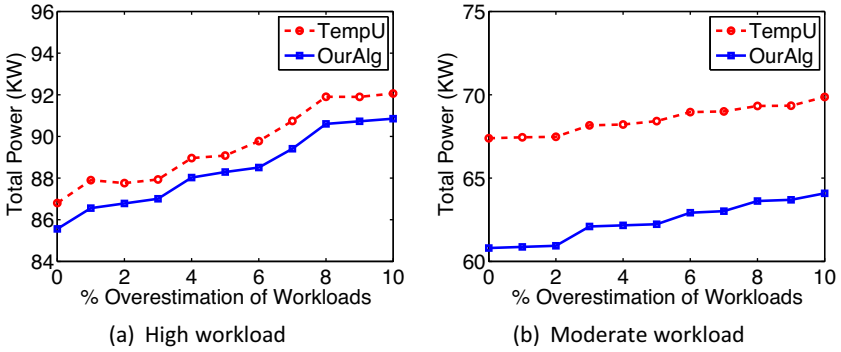


Fig. 12. Performance analysis with workload overestimation.

temperature-reactive approaches, decisions are made *reactively* to avoid server overheating based on the observed real-time temperature. For example, [6] dynamically schedules workloads to "cool" servers based on the instantaneously observed temperature to avoid server overheating; [43] optimally allocates power in a data center for MapReduce workloads by considering the impacts of real-time temperature on the server performance; [5] 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 [44], leading to the development of a less complex yet sufficiently accurate heat transfer model, as shown in (15). For example, without considering delays, [45] maximizes the total capacity of a data center subject to temperature constraint based on this model; [14,26,35] 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 performance (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 [7] combined with dual decomposition [8]. To our best knowledge, our study is the first to apply such technique for solving distributed processing speed selection, VM CPU allocation 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 [28,32,46]).

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

7. Conclusion

We proposed a resource management algorithm, called DREAM, to optimally control the server capacity provisioning, VM CPU allocation and load distribution for minimizing the data center power consumption. By using DREAM, each server can autonomously adjust the *discrete* server processing speed (and hence, power consumption, too), and optimally decide the VM CPU allocation and amount of workloads to process in each VM. 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 average power up to 33%.

Appendix A. Proof of Theorem 1

We provide a proof outline for brevity, while similar proof techniques have also been applied in other contexts such as wireless networks [28,32]. We note first that once the optimal processing speeds are obtained, the optimal VM CPU allocation and load distribution can be easily derived in a distributed manner (by using dual decomposition [46]). As a result, to establish Theorem 1, it is equivalent to prove that the servers' processing speeds selected distributedly will converge to the global optimum following the iterations specified by Algorithm 1.

We denote the servers' processing speeds by $\vec{\mathbf{x}}$. Following the iterations in Algorithm 1, $\vec{\mathbf{x}}$ evolves as a N-dimension Markov chain in which the ith dimension corresponds to server i's processing speed decision and has (K_i+1) possible values. For the ease of presentation, we begin with a 2-server case and denote the state of the Markov chain as S_{x_1,x_2}, where $x_i \in \mathcal{S}_i = \{s_{i,0}, s_{i,1}, \ldots, s_{i,K_i}\}$ for $i = 1, 2$. As shown in line 5 of Algorithm 4.1, only one server is selected to explore a new processing speed at each iteration with equal probability among all servers. Thus, we have

$$\Pr(S_{x_1',x_2'} \mid S_{x_1,x_2}, \text{server 1 updates}) = \begin{cases} \dfrac{\exp\left(\frac{\delta}{\tilde{p}(\vec{\mathbf{x}}')}\right)}{(k_1+1)\left[\exp\left(\frac{\delta}{\tilde{p}(\vec{\mathbf{x}})}\right)+\exp\left(\frac{\delta}{\tilde{p}(\vec{\mathbf{x}}')}\right)\right]}, & \text{if } x_1' \neq = x_1, x_2' = x_2, \\ 0, \text{ if } x_2' \neq = x_2, \\ 1 - \displaystyle\sum_{x_1' \neq = x_1, x_2' = x_2} \dfrac{\exp\left(\frac{\delta}{\tilde{p}(\vec{\mathbf{x}}')}\right)}{(k_1+1)\left[\exp\left(\frac{\delta}{\tilde{p}(\vec{\mathbf{x}})}\right)+\exp\left(\frac{\delta}{\tilde{p}(\vec{\mathbf{x}}')}\right)\right]}, & \text{otherwise}, \end{cases} \tag{20}$$

where $\tilde{p}(\vec{\mathbf{x}})$ is the resulting cost (total power consumption) given $\vec{\mathbf{x}}$ (and the corresponding optimal VM CPU allocation and load distribution decision λ). Thus, the (unconditioned) transition probability can be derived as $\Pr(S_{x_1',x_2'} \mid S_{x_1,x_2}) = \frac{1}{2}\sum_i^2 \Pr(S_{x_1',x_2'} \mid S_{x_1,x_2}, \text{server } i \text{ updates})$, specifying the probability that the state S_{x_1,x_2} transits to state $S_{x_1',x_2'}$. Furthermore, we can show the induced Markov chain is irreducible and aperiodic, implying the convergence to a stationary distribution denoted by Ω. Then, based on balance equation, without loss of generality, we fix the processing speed of one state (e.g., S_{x_1,x_2}) in the Markov chain and derive the following balance equation and some mathematical

manipulations, we obtain the stationary probability distribution for the Markov chain as

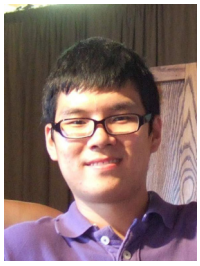
$$\Omega_{\vec{\mathbf{x}}} = \frac{\exp\left(\frac{\delta}{\tilde{p}(\vec{\mathbf{x}})}\right)}{\sum_{\vec{\mathbf{x}}'}\exp\left(\frac{\delta}{\tilde{p}(\vec{\mathbf{x}}')}\right)}, \tag{21}$$

Denote $\vec{\mathbf{x}}^*$ as the optimal capacity provisioning decision that minimizes $\tilde{p}(\vec{\mathbf{x}})$ over all the possible values of $\vec{\mathbf{x}}$, and suppose temporarily that the optimal decision is unique. Letting the temperature $\delta \to \infty$, we can see that at the stationary state, the Markov chain converges to the optimal capacity provisioning decision $\vec{\mathbf{x}}^*$ with a probability of 1. If there exist $M \geq 1$ capacity provisioning decisions, all of which minimize $\tilde{p}(\vec{\mathbf{x}})$, then Algorithm 1 will converge to each of the optimal decisions with a probability of $\frac{1}{M}$, as the temperature $\delta \to \infty$. Finally, the same analysis can be extended to the N-dimensional Markov chain, thereby completing the proof of Theorem 1.

References

[1] B. Hayes, Cloud computing, Commun. ACM 51 (7) (2008) 9–11.
[2] J. Choi, S. Govindan, B. Urgaonkar, A. Sivasubramaniam, Profiling, prediction, and capping of power consumption in consolidated environments, in: MASCOTS, 2008.
[3] H. Lim, A. Kansal, J. Liu, Power budgeting for virtualized data centers, in: USENIX ATC, 2011.
[4] A. Gandhi, M. Harchol-Balter, R. Das, C. Lefurgy, Optimal power allocation in server farms, in: SIGMETRICS, 2009.
[5] J. Chen, R. Tan, Y. Wang, G. Xing, X. Wang, X.-D. Wang, B. Punch, D. Colbry, A high-fidelity temperature distribution forecasting system for data centers, in: RTSS, 2012.
[6] J. Moore, J. Chase, P. Ranganathan, R. Sharma, Making scheduling "cool": temperature-aware workload placement in data centers, in: USENIX ATEC, 2005.
[7] C. Robert, G. Casella, Monte Carlo Statistical Methods, Springer-Verlag, New York, 2004.
[8] D.P. Bertsekas, J.N. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, Athena Scientific, 1989.
[9] Z. Liu, M. Lin, A. Wierman, S.H. Low, L.L. Andrew, Greening geographical load balancing, in: SIGMETRICS, 2011.
[10] M. Lin, Z. Liu, A. Wierman, L.L.H. Andrew, Online algorithms for geographical load balancing, in: IGCC, 2012.
[11] J.R. Lorch, A.J. Smith, PACE: a new approach to dynamic voltage scaling, IEEE Trans. Comput. 53 (2004) 856–869.
[12] R. Urgaonkar, U.C. Kozat, K. Igarashi, M.J. Neely, Dynamic resource allocation and power management in virtualized data centers, in: IEEE/IFIP NOMS, 2010.
[13] S. Ghiasi, T. Keller, F. Rawson, Scheduling for heterogeneous processors in server systems, in: Comput. Front., 2005.
[14] T. Mukherjee, Q. Tang, C. Ziesman, S.K.S. Gupta, P. Cayton, Software architecture for dynamic thermal management in datacenters, in: COMSWARE, 2007.
[15] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, C. Hyser, Renewable and cooling aware workload management for sustainable data centers, in: SIGMETRICS, 2012.
[16] D. Atwood, J. Miner, Reducing Data Center Cost With An Air Economizer, White Paper: Intel Corporation, 2008.
[17] H. Xu, C. Feng, B. Li, Temperature aware workload management in geo-distributed datacenters, in: SIGMETRICS, 2013.
[18] R. Das, J.O. Kephart, J. Lenchner, H. Hamann, Utility-function-driven energy-efficient cooling in data centers, in: Autonomic computing, 2010.
[19] W. Huang, M. Allen-Ware, J.B. Carter, E. Elnozahy, H. Hamann, T. Keller, C. Lefurgy, J. Li, K. Rajamani, J. Rubio, TAPO: thermal-aware power optimization techniques for servers and data centers, in: IGCC, 2011.
[20] J. Moore, J. Chase, P. Ranganathan, R. Sharma, Making scheduling "cool": temperature-aware workload placement in data centers, in: USENIX ATEC, 2005.
[21] L. Rao, X. Liu, L. Xie, W. Liu, Reducing electricity cost: optimization of distributed internet data centers in a multi-electricity-market environment, in: IEEE Infocom, 2010.
[22] K. Le, R. Bianchini, T.D. Nguyen, O. Bilgir, M. Martonosi, Capping the brown energy consumption of internet services at low cost, in: IGCC, 2010.
[23] S. Kundu, R. Rangaswami, A. Gulati, K. Dutta, M. Zhao, Modeling virtualized applications using machine learning techniques, in: VEE, 2012.
[24] L. Wang, J. Xu, M. Zhao, Modeling VM performance interference with fuzzy MIMO model, in: Feedback Computing, 2012.
[25] N.U. Prabhu, Foundations of Queueing Theory, Kluwer Academic Publishers, 1997.
[26] T. Mukherjee, A. Banerjee, G. Varsamopoulos, S.K.S. Gupta, S. Rungta, Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers, Comput. Netw. 53 (17) (2009) 2888–2904.
[27] Google inc. http://www.google.com/about/datacenters/efficiency/external/
[28] Y. Song, C. Zhang, Y. Fang, A game theoretical analysis of joint channel and power allocation in wireless mesh networks, IEEE J. Sel. Areas Commun. 26 (7) (2008) 1149–1159.
[29] A.M. Geoffrion, Generalized benders decomposition, J. Optim. Theory Appl. 10 (4) (1972) 237–260.
[30] H.P. Young, Individual Strategy and Social Structure, Princeton University Press, 1998.
[31] A. Gulati, G. Shanmuganathan, A. Holler, I. Ahmad, Cloud-scale resource management: challenges and techniques, in: HotCloud, 2011.
[32] L. Qian, Y.-J. Zhang, M. Chiang, Distributed nonconvex power control using Gibbs sampling, IEEE Trans. Commun. 60 (12) (2012) 3886–3898.
[33] X. Chen, X. Liu, S. Wang, X.-W. Chang, TailCon: power-minimizing tail percentile control of response time in server clusters, in: SRDS, 2012.
[34] M. Lin, A. Wierman, L.L.H. Andrew, E. Thereska, Dynamic right-sizing for power-proportional data centers, in: Infocom, 2011.
[35] Q. Tang, S.K.S. Gupta, G. Varsamopoulos, Thermal-aware task scheduling for data centers through minimizing heat recirculation, in: CLUSTER, 2007.
[36] Japan meteorological agency (Climate Statistics, 2013).
[37] http://www.timeanddate.com/weather/japan/tokyo/hourly
[38] B. Guenter, N. Jain, C. Williams, Managing cost, performance and reliability tradeoffs for energy-aware server provisioning, in: IEEE Infocom, 2011.
[39] C.-T. Yang, K.-C. Wang, H.-Y. Cheng, C.-T. Kuo, W.C.C. Chu, Green power management with dynamic resource allocation for cloud virtual machines, in: HPCC, 2011.
[40] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, S. Banerjee, Application-aware virtual machine migration in data centers, in: Infocom, 2011.
[41] J. Xu, J.A.B. Fortes, Multi-objective virtual machine placement in virtualized data center environments, in: GREENCOM-CPSCOM, 2010.
[42] H. Liu, C.-Z. Xu, H. Jin, J. Gong, X. Liao, Performance and energy modeling for live migration of virtual machines, in: HPDC, 2011.
[43] S. Li, T. Abdelzaher, M. Yuan, TAPA: temperature aware power allocation in data center with map-reduce, in: IGCC, 2011.
[44] A.H. Beitelmal, C.D. Patel, Thermo-fluids provisioning of a high performance high density data center, Distrib. Parallel Databases 21 (2–3) (2007) 227–238.
[45] K. Mukherjee, S. Khuller, A. Deshpande, Saving on cooling: the thermal scheduling problem, in: SIGMETRICS, 2012.
[46] D.P. Palomar, M. Chiang, Alternative distributed algorithms for network utility maximization: framework and applications, IEEE Trans. Automatic Control 52 (12) (2007) 2254–2269.

Mohammad A. Islam is doing his Ph.D. in Electrical Engineering at Florida International University, Miami, Florida. He received his B.Sc. in Electrical and Electronics Engineering from Bangladesh University of Engineering and Technology in January 2008. He is a member of the Sustainable Computing Group (SCG) at FIU led by Dr. Shaolei Ren. His research interests include data center resource management, cloud computing, and sustainability for IT.



Shaolei Ren received his B.E., M.Phil. and Ph.D. degrees, all in electrical engineering, from Tsinghua University in July 2006, Hong Kong University of Science and Technology in August 2008, and University of California, Los Angeles, in June 2012, respectively. Since August 2012, he has been with School of Computing and Information Sciences, Florida International University, as an Assistant Professor. His research interests include cloud computing, data center resource management, and sustainable computing. He received the Best Paper Award from IEEE International Conference on Communications in 2009 and from International Workshop on Feedback Computing in 2013.

Niki Pissinou has published over two hundred and fifty research papers in peer reviewed journals, conference proceedings and books chapters on networking, telecommunications, distributed systems, mobile computing, security and aspects of nontraditional data management including co-editing over four texts in the area of mobile and wireless networking and systems and over fourteen IEEE and ACM conference volumes. Widely cited in books and research papers, her research has been funded by NSF, DHS, NASA, DOT, DoD, state governments and industry. She has graduated over nineteen Ph.D. students who now hold positions in academia, federal government and

industry. Dr. Pissinou has served as the general and technical program chair on a variety of ACM and IEEE conferences. She also served on hundreds of IEEE and ACM program committees, organizing committees, review panels, advisory boards, editorial boards etc. She has served as an editor of many journals including the IEEE Transactions on Data and Knowledge Engineering. She also has been the founder of many professional forums, including the ACM GIS. Dr. Pissinou has given keynote talks at various events and served as consultant to industry. Her achievements have been recognized by her peers, who have given her several awards and honors, including best paper awards.

A. Hasan Mahmud is a Ph.D. student in the School of Computing and Information Sciences at Florida International University, Miami, Florida. He received his B.Sc. degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in January 2008. Before joining Florida International University, he spent 3:5 years in a leading software company in Bangladesh. He is currently working in the Sustainable Computing Group (SCG) at Florida International University, led by Dr. Shaolei Ren. His current research interests include resource management in cloud computing, capacity provisioning and autoscaling of virtualized resources. He has received the Best Paper Award in 8th International Workshop in Feedback Computing, 2013.

Athanasios V. Vasilakos (M00SM11) is currently a Visiting Professor with the National Technical University of Athens, Greece. He served or is serving as an Editor for many technical journals, such as the IEEE Transactions on Network and Service Management; IEEE Transactions on Information Forensics and Security; IEEE Transactions on Systems, Man, and Cyberneticspart B:Cybernetics; IEEE Transactions on Information Technology in Biomedicine; IEEE Transactions on Cloud Computing, ACM Transactions on Autonomous and Adaptive Systems; the IEEE Journal on Selected Areas in Communications (May 2009, Jan 2011, March 2011 issues). He is also General Chair of the European Alliances for Innovation (http://www.eai.eu).