2015 IEEE International Conference on Data Science and Data Intensive Systems

On Power Management Policies for Data Centers

Zygmunt J. Haas[*†] and Shuyang Gu[†]
*School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
[†]Erik Jonsson School of Engineering and Computer Science, The University of Texas at Dallas, Richardson, TX
{haas, sxg121231}@utdallas.edu

***Abstract*—One of the most important practical and timely operational problem associated with the performance of data centers for cloud computing is energy consumption. The fundamental approach for saving energy in a data center is right-sizing the number of servers; i.e., the determination of the minimum required number of servers to meet the load demand, allowing unnecessary servers to be turned off, so that energy usage can be minimized. The main challenge in designing such right-sizing algorithms is the fact that servers cannot be turned on instantaneously, so typically some estimation of the futuristic load is needed. Of course, the more uncorrelated the load arrival, the less accurate is such estimation. As the problem of right-sizing is NP-complete, a heuristic algorithm is required for practical deployment. In this paper, we first develop an efficient offline right-sizing heuristic, and we demonstrate that its performance is close to optimal. Rather than classifying jobs into a fixed number of types as prior works do, every arriving job is characterized with its own latency tolerance profile. Our offline heuristic, taking advantage of this latency tolerance, attempts to rearranges the jobs' processing times, so that the overall servers' demand is "smoothed". Then, based on the offline algorithm, we design an online algorithm that computes the real-time servers' right-sizing according to the arriving workload. As the key result of this paper, we show that the performance of the online algorithm closely approximates the performance of the offline algorithm, and that both closely approximate the optimal solution. We also demonstrate that the use of our algorithm corresponds to an over 50% operational cost reduction of a data center.**

Keywords—data centers; right-sizing; job profiling; energy-efficient algorithm

I. INTRODUCTION

With the development of cloud computing, the number and the popularity of data centers has increased significantly. However, data centers consume vast amounts of energy, with substantial portion of this energy being used for powering and cooling servers' systems. Thus, saving energy in data centers becomes a very important and timely problem. The fundamental approach for saving energy in a data center is right-sizing the number of servers; i.e., the determination of the minimum required number of servers to meet the load demand, allowing unnecessary servers to be turned off, so that energy usage can be minimized. However, since the load of a data center is unpredictable, and often characterized by low temporal correlation, right-sizing is a challenging problem.

Although there have been a lot of research efforts in studying the right-sizing problem (e.g., [1] – [6], [12], [13]), the characteristics of the applications and the specific attributes of the jobs handled by the data centers have not been adequately taken into account. Consequently, a goal of this paper is to develop an *online* algorithm that makes use of the latency tolerance of jobs as to attempt to smooth the servers' demand over time, so that service can be provided with smaller number of servers, thus saving overall data center's energy.

The rest of this paper is organized as follows. Section II presents our models of the servers and the workload, which are used to formulate the energy optimization problem. Section III solves the offline optimization problem by taking advantage of jobs' latency tolerance. Using the offline algorithm, Section IV describes the design of our online algorithm. Section V presents the performance analysis, and Section VI concludes the paper.

II. MODEL FORMULATION

A. Workload Model

We assume that time is divided into time-slots and that the total "experiment" time lasts T time-slots, {1, …. , T}. We further assume that all events occur at the time-slots boundaries; i.e., let $t \in \{1, ..., T\}$ stand for a scheduling instant. Note that the length of each time-slot is assumed to be on the order of the time needed to activate a server (i.e., time needed to power on a server – see below). Also, let the average arrival rate of requests to the data center during a time-slot t be λ^t.

Each job $j \in \{1, \dots, |J|\}$ (where J is the set of all the jobs in the system) is characterized with three parameters: the arrival time τ_j, the amount of workload w_j(in slots), and the lifetime δ_j(in slots), where the latter is the maximum time that the job is allowed to spend in the data center.

B. Server Model

We consider homogeneous servers model. Each server may be in one of the three states, which are: *on*, *off*, and *switching*. This three-state model was adopted by many papers addressing the problem of data centers right-sizing (e.g., [1], [12]).

- The *on* state: A server is powered and ready to process jobs, and it expends E_{slot} energy per time-slot no matter whether the server actually processes a job or not.
- The *off* state: A server is not powered, it cannot process any job, and it does not consume any energy.
- The *switching* state: A server transitions from the *off* to the *on* state. The server will remain in the *switching* state for t_{on} time-slots, at the end of which the server will enter the *on* state. While in the *switching* state, the server is powered and consumes E_{on} energy per time-slot, but is not available for jobs processing.

We make an assumption that going from the state *on* to the state *off* costs no extra energy and that such a transition can occur immediately while the unfinished jobs on that server can be transferred to other servers with no cost. Also, a server can be turned off while in the *switching* state without any extra energy, and such a transition occurs instantaneously. Finally, as it is common in this field (e.g., [1], [5], [12]), we assume that a server can serve only one job at a time.

C. The Energy Optimization Problem

We label $S^{on}(t)$ and $S^{+}(t)$ as the number of servers that are in the *on* state and that are in the *switching* state, respectively, at the beginning of the time-slot t. $W(t)$ is the number of jobs that will be processed during the time-slot t. $P_j(t)$ is a binary variable, where $P_j(t)$ =1 if the job j is processed during the time-slot t, and $P_j(t)$= 0 otherwise. N is the total number of servers in the data center.

Given the above models, the goal of our algorithm is to decide when to switch servers on or off, as to minimize the

978-1-5090-0214-6/15 $31.00 © 2015 IEEE
DOI 10.1109/DSDIS.2015.82

total energy cost during the time [*1, …, T*], while meeting all jobs' latency requirements. Specifically, the optimization problem is formulated as follows:

minimize

$$\sum_{t=1}^{T}[E_{slot} \cdot S^{on}(t) + E_{on} \cdot S^{+}(t)] \quad (1)$$

subject to

$$S^{on}(t) \geq W(t), t = 1,2, \ldots T. \quad (2)$$

$$W(t) = \sum_{j=1}^{|J|} P_j(t), t = 1,2, \ldots T. \quad (3)$$

$$\forall j: \sum_{t=\tau_j}^{\tau_j+\delta_j-1} P_j(t) = w_j \quad (4)$$

$$\forall j: P_j(t) = 0, 1 \leq t < \tau_j \text{ or } t > \tau_j + \delta_j - 1 \quad (5)$$

$$S^{on}(t) + S^{+}(t) \leq N, t = 1,2, \ldots T \quad (6)$$

III. The Offline Heuristic

The *offline* algorithm assumes that all the information (i.e., $\forall j$, τ_j, w_j, and δ_j) is known before the "experiment" starts (i.e., at time t=1). We can then look at the task of the algorithm to distribute 1's in a matrix of $|J| \times T$, such that for each job (in each row), there are exactly w_j 1's and that all the 1's are within the time-slots $(\tau_j, \tau_j + \delta_j - 1)$, where a 1 in the ($j$, t) place corresponds to $P_j(t) = 1$. (Such a matrix is referred to as a *feasible solution.*) In other words, the problem is how to distribute the workload w_j in the δ_j time-slots (starting from time-slot τ_j) for each job j. Each such an assignment of jobs, $P_j(t)$, requires a certain number of servers to be on, and, thus, corresponds to some required amount of energy. Knowing this required number of servers in each time-slot allows us to determine when to turn on/off servers, so that, on one hand, the number of servers in the *on* state is minimized, yet, on the other hand, the number of servers in the *on* state matches the $\{P_j(t)\}$ demand. The optimal solution is, then selecting the smallest energy solution from among all the *feasible solutions.*

A. Workload Distribution Problem

As discussed above, we can represent the assignment $P_j(t)$ for all the jobs as a matrix, with a corresponding graph of $W(t)$, which depicts the number of required *on* servers. For example, assume three jobs: $\tau_1 = 1$, $\tau_2 = 2$, $\tau_3 = 2$, $w_1 = 1$, $w_2 = 1$, $w_3 = 2$, $\delta_1 = 3$, $\delta_2 = 2$, $\delta_1 = 4$. Each job is represented by a row in the "assignment graph" (e.g., Fig 1(a)), and if $P_j(t) = 1$, then the corresponding time-slot t is marked with '1', where the number of '1's of a job must be equal to its workload, and also every '1' must be within the grey rectangle which represents the lifetime of the job. Fig. 1 shows the optimal assignment for these three jobs, as well as the server demand curve for the assignment.

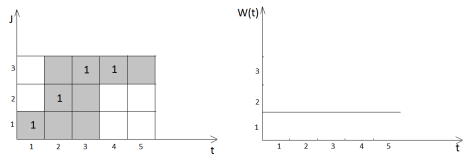

Fig. 1: (a) Assignment graph; (b) Server demand curve.

The area below the $W(t)$ curve is the energy needed for processing all the jobs, this area is the same for every possible assignment, because it equals to $\sum_{j=1}^{|J|} w_j$. However, the actual $W(t)$ curve determines the number of servers that needs to be turn on (some of which have to be turned on ahead of time, in order to satisfy the $W(t)$demand). More particularly, it is the rising edges of the $W(t)$ curve that determines when and how many servers need to be turned on. Switching on servers costs extra energy, so to minimize the total energy we should attempt to minimize the number of these rising edges. We accomplish this by 'flattening' the server-demand curve, using the fact that for any job, any assignment is valid as long as it is made within the $(\tau_j, \tau_j + \delta_j - 1)$ interval.

A heuristic way of flattening the curve is first to assign all the possible time-slots within the $(\tau_j, \tau_j + \delta_j - 1)$ interval; i.e., letting $P_j(t) = 1, t = \tau_j, \ldots, \tau_j + \delta_j - 1$, to get a redundant server demand curve, and then repeatedly remove these redundant assignments in an attempt to flatten the peaks of the curve until the number of the assigned time-slots is equal to the workload for each jobs. Denote the number of assigned time-slots as r_j, $r_j = \sum_{t=\tau_j}^{\tau_j+\delta_j-1} P_j(t)$, initially $r_j = \delta_j$. Also we should consider the initial state of the servers at the beginning of the experiment and the fact that because servers cannot be turned on instantaneously, not every server demand curve is feasible. For example, we have $S^{on}(1) = 1$ and $S^{+}(1) = 0$, meaning that one server is on and no server is being transitioning to the on state at the start of the experiment. But there can be as many servers on as needed after t_{on} time-slots by turning on the servers at the beginning of the time-slot 1. We can represent this restriction as a function $l(t)$. We need to modify $P_j(t)$ so that for $t \in \{1, \ldots, t_{on}\}, W(t) \leq l(t)$. Then we flatten the peak of W(t) curve by removing the assigned time-slots for the job with most redundancy (the job for which $(r_j - w_j)/w_j$ is the largest), update $W(t)$ and flatten it repetitiously until for all jobs $r_j = w_j$.

The execution of our heuristic is demonstrated with our simple example. Fig. 2 shows the initial assignment. Fig. 3 shows the assignment after a single iteration. The red curve in the Fig. 3(b) shows the maximum number of the servers that can be on at any time-slot, taking into consideration the initial conditions and $t_{on} = 2$. The process of repetitious "flattening" the server demand curve is shown in Figs. 3-6.

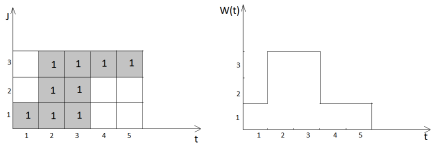

Fig. 2: (a) Initial assignment graph; (b) Server demand curve.

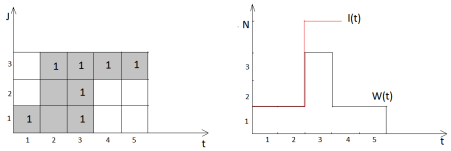

Fig. 3: (a) Assignment graph conform to server limit; (b) Server demand curve and server limit.

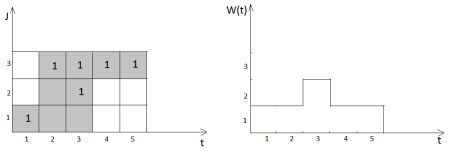

Fig. 4: (a) Next iteration of assignment graph; (b) Server demand curve.

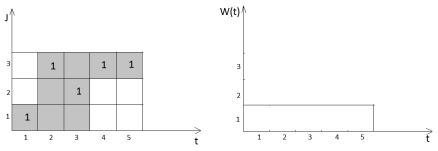

Fig. 5: (a) Next iteration of assignment graph; (b) Server demand curve.

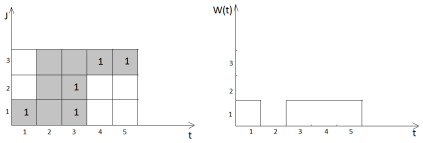

Fig. 6: (a) Final assignment graph; (b) Server demand curve.

Next, we present the offline heuristic in pseudo-language:

The offline heuristic algorithm

1: **for** each $j \in J$ **do**
2: **for** $t = \tau_j$ to $\tau_j + \delta_j - 1$ **do**
3: $P_j(t) = 1$
4: **end for**
5: **end for**
6: $R_n = \{x | x = n, x \in R\}$
7: $l(1) = S^{on}(1)$
8: $l(t) = |R_{t-1}| + S^{on}(1), t = 2, \dots, t_{on}$
9: $k(t) = l(t) - W(t), t = 1, \dots, t_{on}$
10: **for** t= 1 to t_{on} **do**
11: **if** $k(t) < 0$ **then**
12: $J_t = \{j | P_j(t) = 1\}$
13: **for** $i = 1$ to $-k(t)$ **do**
14: set $P_j(t) = 0$ for $j \in J_t$ and $j = \arg max[(r_j - w_j)/w_j]$
15: update $J_t = \{j | P_j(t) = 1\}$
16: **end for**
17: **end if**
18: **end for**
19: **while** $\exists j$ s.t. $\sum_{t=\tau_j}^{\tau_j+\delta_j-1} P_j(t) < w_j$ **do**
20: $J = J \backslash \{j\}$
21: $n = n + 1$
22: **end while**
23: **while** $\exists j, s.t. \sum P_j(t) > w_j$ **do**
24: **for** t = 1 to T **do**
25: $t_m = \arg\max [W(t)]$
26: $J_m = \{j | P_j(t_m) = 1\}$
27: **for** j = 1 to $|J_m|$ **do**
28: **if** $\exists\, j \in J_m$, $j = \arg max[(r_j - w_j)/w_j]$
29: and $[(r_j - w_j)/w_j > 0$ **do**
30: set $P_j(t) = 0$
31: **return**
32: **end if**
33: **end for**
34: **end for**
35: **end while**
36: **while** $\exists t$ s.t. $W(t) > N$ **do**
37: $J_l = \{j | P_j(t_m) = 1\}$
38: $j = \arg\max [w_j]$, $j \in J_l$
39: $\forall t \in \{1, \dots T\}, P_j(t) = 0$
40: $J = J \setminus \{j\}$
41: $n = n + 1$
42: **end while**

Lines 1-5 of this algorithm assign for each job all the possible time-slots, starting from the job's arrival time-slot. Lines 6-18 calculate the server limit and then improve $W(t)$ to match the server limit $l(t)$ from time-slot 1 to time-slot t_{on}. R is the set that contains the remaining switching time of all switching servers. J_t is the job set that contains all the jobs that were to be processed in time-slot t. Then the algorithm checks in lines 19-22 if there is any job with assigned time less than its workload. If so, the algorithm drops that job and updates the jobs set J and $P_j(t)$; it also updates the number of lost jobs n.

Lines 23-35 represent the process of flattening the curve $W(t)$ by removing its peak repeatedly: finding the time-slot with the heaviest load and, for all the jobs that have been assigned in this time-slot, finding the one with the biggest redundancy, removing its assignment and updating $W(t)$. At the end of this part, the assignment has no remaining redundancies. In lines 36-42, the algorithm checks whether there is any time-slot that has workload larger than the number of servers, deletes excess jobs and updates the assignment and the jobs set.

At the end of the algorithm's processing, the final assignment $P_j(t)$ and the server demand function $W(t), t = 1,2, \dots, T. j = 1,2 \dots |J|$ are obtained. From this final assignment, the decisions to turn servers on or off are performed for each time-slot $t \in \{1, \dots, T\}$, so that the energy cost is minimized (see the next subsection).

B. The Server Decision Problem

At the beginning of each time-slot, we want to make the necessary decisions to turn servers on or off, so that the total energy cost is minimized, as long as the server demand could be satisfied.

After we solved the workload distribution problem above, we obtained the function $W(t)$, which is the least number of servers that should be in the *on* state in time-slot t. Using this function, we can calculate the (earliest) time until when any number of servers is needed to be on. Denote the earliest time that x servers are needed to be on as $W^{-1}(x)$. Note that the decision of keeping a number of servers on or transitioning them to the *switching* state depends not only on $W^{-1}(x)$, but also on the relationship between the values of t_{on}, E_{slot}, and E_{on}, as discussed in the following.

Energy Saving Analysis for the case $E_{on} \geq E_{slot}$

In what follows, i represents the current time-slot in which the decision is to be make, and t_S is the next time-slot in which a currently unused server is needed again.

Case 1: $t_S - i < t_{on} + 1$. That is to say, there is not enough time to turn the server off and then turn it on again to meet the demand, then this server should be kept on anyway.

Case 2: $t_S - i \geq t_{on} + 1$, $[t_S - i] \cdot E_{slot} \leq t_{on} \cdot E_{on}$. In this case, there is enough time to turn the server off and then turn it on again to meet the demand, however, keeping it on costs less energy. Therefore, this server should be kept on.

Case 3: $t_S - i \geq t_{on} + 1$, $[t_S - i] \cdot E_{slot} > t_{on} \cdot E_{on}$. In this case, there is enough time to turn the server off and then turn it on again to meet the load, and also turning it off and then turning it on again costs less energy, so this server should be turned off.

The offline decision algorithm for $E_{on} \geq E_{slot}$

1: $t_{max} = \max [floor\left(\frac{E_{on}}{E_{slot}}\right) \cdot t_{on}, t_{on} + 1]$
2: $d_{max} = \max[W(t)], t = i, \dots, i + t_{max} - 1$

3: **if** $l(i) \geq d_{max}$ **then**
4: $S^{on}(i) = d_{max}$
5: $S^{+}(i) = 0$
6: **else if** $l(i) < d_{max} \leq l(i) + S^{+}(i)$
7: $S^{+}(i) = l(i) + S^{+}(i) - d_{max}$
8: **else if** $d_{max} > l(i) + S^{+}(\mathrm{i})$ **then**
9: $S^{+}(i) = S^{+}(i) + min\{N - l(i) - S^{+}(i), d_{max} - l(i) - S^{+}(i)\}$
10: **end if**
11: **end if**
12: **end if**

The algorithm first calculates the longest time a server can be idle t_{max}. Then calculate the maximum demand from time-slot i to time-slot i+$t_{max} - 1$, denoted as d_{max}. The servers decision then would be made based on the relationship between d_{max}, $l(i)$, and $S^{+}(i)$.

Energy Saving Analysis for the case $E_{on} < E_{slot}$

When E_{on} is smaller than E_{slot}, turning on servers just before they are needed will save energy.

The offline decision algorithm for $E_{on} < E_{slot}$

1: $d_{diff} = \min[l(x) - W(x)], x \in [i, i + t_{on} - 1]$
2: $d_{max} = \max[W(x)], x = i \ldots, i + t_{on}$
3: $S^{on}(i) = S^{on}(i) - d_{diff}$
4: **if** $d_{max} - l(\mathrm{i}) + d_{diff} \leq S^{+}(i)$ **then**
5: $S^{+}(i) = d_{max} - l(\mathrm{i}) + d_{diff}$
6: **else** $S^{+}(i) = S^{+}(i) + \min\{N - l(\mathrm{i}) + d_{diff} - S^{+}(i), d_{max} - l(\mathrm{i}) + d_{diff} - S^{+}(i)\}$
7: **end if**

The algorithm first calculates the minimum difference between servers limit and servers demand, $d_{diff} = min[l(x) - W(x)], x \in \{i, i + t_{on} - 1\}$ and also calculates the maximum demand d_{max} from the time-slot i to i+t_{on}. Then the servers' decisions are made based on the relationship between d_{max}, d_{diff}, $l(i)$, and $S^{+}(i)$.

IV. The Online Heuristic

To come up with an online algorithm, we use the offline heuristic algorithm in each time-slot with the existing workload (past arrivals) and an anticipated average future workload to obtain an "optimal" assignment. Then based on this assignment, we make decisions for the servers in the current time-slot. For simplicity, we choose the average workload as the anticipated future load; this choice also tends to optimize the overall performance of our scheme. Note that unlike the offline algorithm, the online algorithm does not assume the overall workload for future time is known, all that affect its decision is the existed jobs and estimated workload while the decision will be adjusted according to the real arrived jobs whenever necessary.

A. Job Distribution Algorithm

The main difference between the online heuristic and the offline heuristic is that the online algorithm will be used at the beginning of every time-slot to determine which jobs would be processed during the current time-slot. The proposed online heuristic algorithm uses the average workload and lifetime as the predicted future job arrivals. Then for the combination of the existing jobs and the predicted future jobs, the demand curve flattening is applied, while considering the current server limitation.

Another difference of the online algorithm is that we do not drop any job that are to be processed in the future, because the actual load in the future may be actually lower than the average load that is used to calculate the current assignment. Consequently, the heuristic offline algorithm is modified as following to come up with the online algorithm. (Assume that the current time for the algorithm is the time-slot *i*.)

The online heuristic algorithm

1: **for** each $j \in J$ **do**
2: **for** $t = \tau_j$ to $\tau_j + \delta_j - 1$ **do**
3: $P_j(t) = 1$
4: **end for**
5: **end for**
6: $\mathrm{R_n} = \{\mathrm{x} | \mathrm{x} = \mathrm{n}, \mathrm{x} \in \mathrm{R}\}$
7: $l(i) = S^{on}(i)$
8: $l(i + t) = |R_t| + S^{on}(1), t = 1, \ldots, t_{on} - 1$
9: $k(t) = l(t) - W(t), t = i, \ldots, i + t_{on} - 1$
10: **if** $k(i) < 0$ **then**
11: $J_t = \{j | P_j(i) = 1\}$
12: **for** $n = 1$ to $W(i) - l(i)$ **do**
13: set $P_j(t) = 0$ for $j \in J_t$ and $j = \arg max[(r_j - w_j)/w_j]$
14: update $J_t = \{j | P_j(i) = 1\}$
15: **end for**
16: **end if**
17: **while** $\exists j$ s.t. $\sum_{t=\tau_j}^{\tau_j + \delta_j - 1} P_j(t) < w_j$ **do**
18: $J = J \backslash \{j\}$
19: $n = n + 1$
20: **end while**
21: **while** $\exists j, s.t. \sum P_j(t) > w_j$ **do**
22: **for** t = i to T **do**
23: $t_m = \arg\max [W(t)]$
24: $J_m = \{j | P_j(t_m) = 1\}$
25: **for** j = 1 to $|J_m|$ **do**
26: **if** $\exists\, j \in J_m, j = \arg max[(r_j - w_j)/w_j]$ and $[(r_j - w_j)/w_j > 0$ **do**
27: set $P_j(t) = 0$
28: **return**
29: **end if**
30: **end for**
31: **end for**
32: **end while**

Lines 1-5 of the algorithm assign all possible time-slots for each job to obtain a redundant assignment. From this assignment, we obtain the workload function $W(t)$. Lines 6-16 calculate the server limit and then improve the $W(i)$ function to match the server restriction function $l(i)$ for current time-slot. After that, the algorithm checks if there is any job that has been assigned less time than its workload (line 17-20), and if so drops that job, updating the number of lost jobs. Also, $P_j(t)$ is updated accordingly. Lines 21-32 flatten the curve $W(t)$ by repeatedly smoothing the peaks of the curve.

From the final assignment, the decisions for the servers at each time-slot $t \in \{1, ..., T\}$ are derived, so that the total energy cost is minimized.

B. Server Decision Algorithm

The decision algorithm for the case of $E_{on} \geq E_{slot}$ is the same as the offline decision algorithm.

The online decision algorithm for $E_{on} < E_{slot}$

1: $d_{diff} = \min[l(x) - W(x)], x \in [i, i + t_{on} - 1]$
2: $d_{max} = \max[W(x)], x = i \ldots, i + t_{on}$

3: **if** $d_{diff} \geq 0$ **then**
4: $S^{on}(i) = S^{on}(i) - d_{diff}$
5: **end if**
6: **if** $d_{max} - l(\mathrm{i}) + d_{diff} \leq S^{+}(i)$ and $d_{diff} \geq 0$ **then**
7: $S^{+}(i) = d_{max} - l(\mathrm{i}) + d_{diff}$
8: **else if** $d_{max} - l(\mathrm{i}) + d_{diff} > S^{+}(i)$ and $d_{diff} \geq 0$ **then**
9: $S^{+}(i) = S^{+}(i) + \min\{N - l(\mathrm{i}) + d_{diff} - S^{+}(i), d_{max} - l(\mathrm{i}) + d_{diff} - S^{+}(i)\}$
10: **else if** $d_{diff} < 0$ **then**
11: $S^{+}(i) = S^{+}(i) + \min\{-d_{diff}, (d_{max} - l(\mathrm{i}) - \mathrm{S+i}\}$
12: **end if**
13: **end if**
14: **end if**

Similar to the offline decision algorithm, the online decision algorithm for the case of $E_{on} < E_{slot}$ (lines 1-2) first calculates the minimum difference d_{diff} between the servers limit curve and the servers demand curve, as well as the maximum demand d_{max} from time-slot i to time-slot $i+t_{on}$. Then the decisions are made for servers that need to be on based on whether d_{diff} is greater than zero (lines 3-5). The decisions for servers that need to be switching depend on d_{diff} and the relationship between $[d_{max} - l(i) + d_{diff}]$ and $S^{+}(i)$ (lines 6-14).

V. Performance Evaluation

To evaluate and compare the performance of the proposed algorithms, we define a utility function that takes into account the energy used to process the jobs and the "cost" due to lost jobs. The rationale for this utility function is that there is a clear tradeoff between the energy consumption and the job completion rate; e.g., an algorithm can use very little energy, but at the expense of low job completion rate. More specifically, we formulate the total cost, C, as follows:

$$C = E_{total} + \alpha D \quad (7)$$

where E_{total} is total energy cost, D is the number of lost jobs (i.e., failed to complete processing within their lifetimes), and α is a scaling factor that expresses the relative importance of energy E_{total} vs. lost jobs D. For simplicity, we assume that α is equal for all jobs; however, if the jobs have different level of importance, equation (7) could be adjusted to express the relative importance of a lost job vs. the cost of energy.

We simulated and compared the performance of the proposed two heuristics (offline and online), as well as a "non-scheduling algorithm". The non-scheduling algorithm is an online algorithm with the strategy that every job is processed as soon as it arrives at the data center. In this algorithm, we assume that the service discipline is first come first serve (FCFS) and that the waiting queue is infinite (the job loss is entirely due to missing their deadlines). Thus, any job whose execution is not completed before its deadline is removed from the system. The arrival jobs are simulated by Poisson random variables with the average arriving rate of λ. The workload and lifetime of each jobs are simulated in the same way, with the average workload $\bar{w}$ and the average lifetime $\bar{\delta}$.

A. Comparison with the optimal offline algorithm

The problem of finding the optimal assignment is, of course, NP-complete. However, comparison of our heuristics with the optimal solution, even for a small-size cases, can provide a good indication on the well our heuristics perform.

For comparison purposes, we designed a "brute-force" scheme based on binary integer programming to calculate the optimal offline assignment according to formula (1)-(6) and implemented it in MATLAB for cases of up to 20 time-slots. The results of the comparison are shown in Fig.7. The results in the figure demonstrate that:

(1) Our offline algorithm performs close to our online algorithm
(2) Both our heuristics perform close to the optimal solution
(3) Both our algorithms reduce the total cost by more than 50%, compared to the non-scheduled cases.

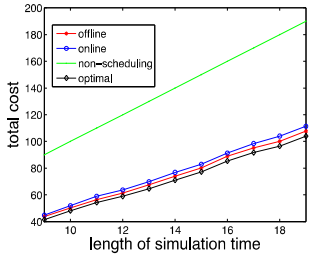

Fig. 7. Total cost vs. simulating time with different algorithms. (E_{slot} = 1, E_{on} = 1.5, N=30, λ = 2, α = 5, $\bar{w}$ = 3, $\bar{\delta}$ = 6.)

Furthermore, Table 1 shows the difference in total cost

	t_s =9	t_s =10	t_s =11	t_s =12	t_s =13	t_s =14
$e_{offline}$%	5.26	4.16	4.13	3.98	4.53	4.07
e_{online}%	7.9	7.9	8.48	8.05	8.12	8.05
	t_s =15	t_s =16	t_s =17	t_s =18	t_s =19	
$e_{offline}$%	3.67	4.2	3.85	3.79	3.57	
e_{online}%	7.41	6.95	7.37	7.82	7.12	

TABLE 1. ERROR OF THE HEURISTICS

(referred to as "relative error") of the two heuristics as compared with the optimal algorithm, demonstrating that the relative error (in %) decreases with larger simulation cases. These results suggest that the error of our heuristics continues to be small for large, real-life cases.

B. Comparison of the online with the offline heuristics

We compared the performance between the online heuristic

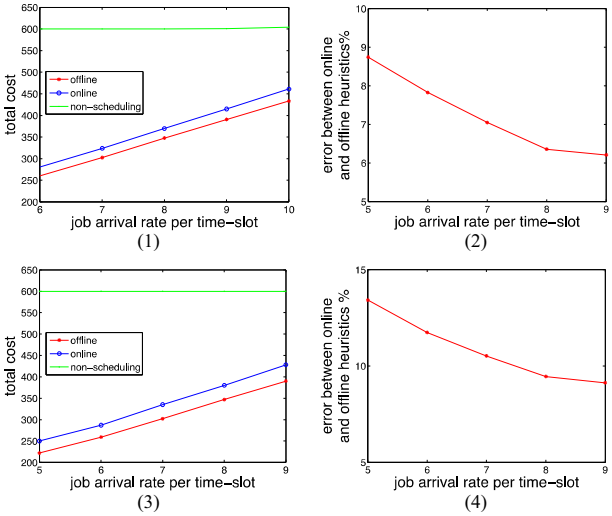

Figure 8: (1) Total cost vs. λ (α = 5). (2) Error between online and offline heuristics. (3) Total cost vs. λ (α = 10). (4) Error between online and offline heuristics. (E_{slot} = 1, E_{on} = 1.5, N=30, t_s = 20, $\bar{w}$ = 3, $\bar{\delta}$ = 6.)

and offline heuristic against λ , with the results shown in Fig.8. The difference between the heuristics decreases with an increase in λ, as shown in Fig.8(2). Comparing Fig.8(1) with Fig.8(3), we can observe the effect of increasing α from $\alpha = 5$ to $\alpha = 10$. The results demonstrate that dropping jobs has a significant impact on the total cost. Furthermore, we observe that larger λ decreases the difference between the two heuristics, suggesting that the online algorithm performs better in heavy-loaded systems.

The impact of the ratio E_{on}/E_{slot} is shown in Fig.9(1). E_{on}/E_{slot} is an indication of how much more expensive it is to switch a server off and then on, rather than leaving the server on and idle. Both the online and the offline algorithms involve switching off/on servers to save energy, so the total cost increases with (E_{on}/E_{slot}). However, when E_{on}/E_{slot} becomes large enough, the advantage of right-sizing becomes less pronounced, because E_{on} is so large that we do not bother switching servers off.

As shown in Fig.9(2), there is no significant impact caused by changing the prediction window for the online algorithm.

Fig.9(3) shows the total cost vs. setup time t_{on}. Both the online and offline heuristic results increase with t_{on}, since the larger t_{on} is, the more jobs would be dropped when there is a temporary increase in load. Moreover, when the system load is smaller than the expected load for short time durations, the

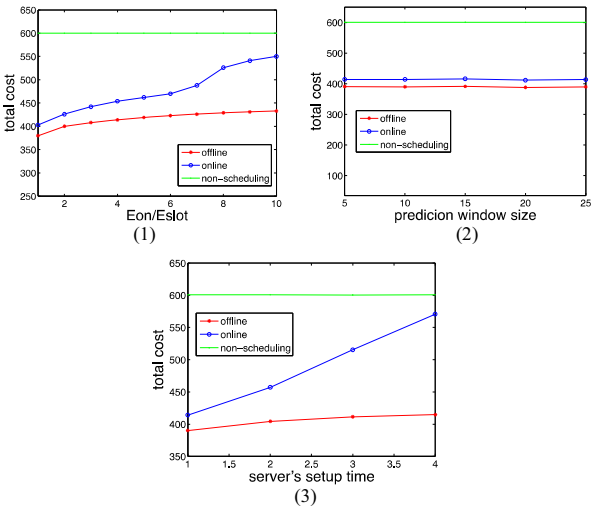

Fig.9 (1) Total cost vs. E_{on}/E_{slot}. (E_{slot} = 1, N=30, t_{on}=1, t_s = 20, λ = 9, α = 5.) (2) Total cost C vs. prediction window t_s. (E_{slot} = 1, E_{on} = 1.5, t_{on}=1, N=30, λ = 9 α = 5.) (3) Total cost C vs. setup time t_{on}. (E_{slot} = 1, E_{on} = 1.5, t_s = 20, N=30, λ = 9, α = 5, $\bar{w} = 3$, $\bar{\delta} = 6$)

algorithms tend to keep servers on in case the load increases within the t_{on} time-slots. However, the online algorithm is more affected by t_{on} than the offline algorithm due to the fact that the online algorithm relies on predicting the future load, and that inaccuracy of such predictions become more significant when the t_{on} increases.

C. Comparison with another scheduling scheme

In order to test the performance of our algorithms, we also compare our heuristics with the algorithm proposed in reference [12], denoted as "Lin's algorithm" below.

Lin's algorithm

In Lin's paper, there are two types of optimization in terms of QoS requirement, the one that considers service with hard QoS constraints is similar to our job's lifetime assumption. That optimization is formulated as:

$$\text{minimize} \quad \sum_{t=1}^{T} x_t\, p_t e\left(\frac{\lambda_t}{x_t}\right) + \beta \sum_{t=1}^{T} (x_t - x_{t-1})^+$$
$$\text{subject to} \quad x_t \geq \lambda_t \text{ and } d\left(\frac{\lambda_t}{x_t}\right) \leq D_t$$

The duration of time-slot of Lin's model $t \in \{0,1,\ldots T\}$ matches the timescale at which the data center can adjust its capacity, which is similar to t_{on} in our model. p_t is the electricity price at time-slot *t*. x_t is the number of active servers at time-slot *t*. β is the cost to turn on a server. λ_t is the mean arrival rate at time-slot *t*. The power cost function is $e(\lambda) = e_0 + e_1\lambda$. The average delay is $d(\lambda) = 1/(\mu - \lambda)$. Since the service rate of a server μ is normalized to 1, $\lambda = \lambda_t/x_t \leq 1$.

Since we don't consider the electricity price impact and load dependent energy consumption, we set $e_1 = 0$ in our simulations, thus making the energy consumption load-independent. Also, we set $p_t = 1$, so that the electricity price does not affect the optimization. Then the optimization model of Lin's paper reduces to:

$$\text{minimize} \quad \sum_{t=1}^{T} x_t\, e_0 + \beta \sum_{t=1}^{T} (x_t - x_{t-1})^+$$
$$\text{subject to} \quad x_t \geq \lambda_t \text{ and } d\left(\frac{\lambda_t}{x_t}\right) \leq D_t$$

The total cost of Lin's optimization consists of the energy consumed by active servers and switching servers. The minimum number of active servers at time *t* is constrained by the condition $x_t \geq \lambda_t$ and $d(\frac{\lambda_t}{x_t}) \leq D_t$. In order to find out the right timing to turn on/off servers to minimize the total energy, they introduce the lazy capacity provisioning algorithm. This algorithm is motivated by the structure of the optimal offline solution.

The main difference between our algorithm and Lin's algorithm

The optimization object of Lin's algorithm is the same as ours – to save the energy cost by adjusting the number of active servers. One of the main differences between Lin's algorithms and our heuristics is that the number of active servers in Lin's algorithms is determined by the mean arrival rate while taking the mean delay into account. The mean delay is calculated by the theoretical queuing model. To achieve a desired average delay, the number of servers must be large enough, so that the arrival rate for each server guarantees the average delay, D_t : $d(\lambda_t) = 1/(1 - \frac{\lambda_t}{x_t}) \leq D_t \Rightarrow x_t \geq \lambda_t/(1 - \frac{1}{D_t})$. For our algorithm, we treat each job individually, we assign the processing time to each job based on its workload and lifetime, while considering the overall workload. Jobs in our model can be suspended and transferred to different servers to resume. While in Lin's model jobs allocated to a server form a queue, jobs cannot change server, and when a job is being processed the service cannot be turned off.

In Lin's algorithm, the decision of the number of servers is made every 10-min time-slot, with the assumption that the job interarrival time and the response time are much shorter than the time-slot. In order to compare our algorithm with Lin's algorithm, we divide each 10-min time-slot into many shorter time-slots (we refer to those as "short time-slots") that match the workload of jobs. We let the average workload be 2 short time-slots long, the average lifetime be 6 short time-slots long. We set $\beta = 6, e_0 = 1$ for Lin's algorithm, while $E_{on} = 6, E_{slot} = 1$ for our algorithm.

Comparison of the offline algorithms

We run simulations comparing our and Lin's offline algorithms. In the following graphs, the red curve represents the number of servers over time as calculated by Lin's algorithm (the model of services with hard QoS constraints),

and the blue curve represents the overall workload using our algorithm. As can be seen, the number of servers using our algorithm is much less than the number of servers of Lin's algorithm. Furthermore, no jobs are dropped using our algorithm, while a job is going to spend more time than the lifetime (average delay) with a probability of 0.37 in Lin's model.

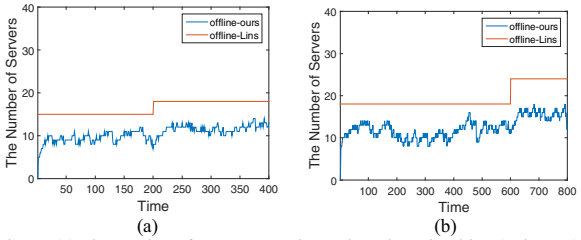

Fig.10 (a) The number of servers over time using Lin's algorithm (red curve) and overall workload (blue curve) using our algorithm with a simple workload trace [5 6]. (b) The number of servers over time using Lin's algorithm (red curve) and overall workload (blue curve) using our algorithm with a simple workload trace [6 5 6 8].

Since, given the arriving rate at each time-slot, the cost difference between our and Lin's offline algorithms is independent between time-slots, we simulated the comparison only for a limited number of time-slots. (The workload trace [5 6] used in the simulation Fig.10(a) means the arrival rate of the first time-slot is 5 jobs per short time-slot, the arrival rate of the second time-slot is 6 jobs per short time-slot, each time-slot is divided into 200 short time-slots. The workload trace [6 5 6 8] is of the same meaning.)

Comparison of the online algorithms

In order to tradeoff the switching cost and the lost job cost, we modified our online decision algorithm as: switch on more servers only when the duration of heavy load is longer than a threshold.

The online decision algorithm for $E_{on} \geq E_{slot}$

1: $t_{max} = \max\left[floor\left(\frac{E_{on}}{E_{slot}}\right) \cdot t_{on}, t_{on} + 1\right]$
2: $d_{max} = \max[W(t)], t = i, \ldots, i + t_{max} - 1$
3: $d_{tOn} = \max[W(t)], t = i, \ldots, i + t_{on}$
4: **if** $l(i) \geq d_{max}$ **then**
5: $S^{on}(i) = d_{max}$
6: $S^{+}(i) = 0$
7: **end**
8: **if** $l(i) < d_{tOn} \leq l(i) + S^{+}(i)$
9: $S^{+}(i) = d_{tOn} - l(i)$
10: **end**
11: **if** $d_{tOn} > l(i) + S^{+}(i)$
12: **for** n = 1: $min\{N - l(i) - S^{+}(i), d_{tOn} - l(i) - S^{+}(i)\}$
13: **if** R[$l(i)$ + S^{+}(i)+n] ≥ threshold
14: $S^{+}(i)++$
15: **end if**
16: **end for**
17: **end if**

The following graph shows the comparison result of online algorithms with a simple workload trace. ([1 2 2 4], the arrival rate of the first time-slot is 1 jobs per short time-slot, the arrival rate of the second time-slot is 2 jobs per short time-slot, the arrival rate of the third time-slot is 2 jobs per short time-slot, the arrival rate of the fourth time-slot is 4 jobs per short time-slot.) Lin's online algorithm Lazy Capacity Provisioning (LCP(w)) assumes that at time τ, LCP(w) knows the arrival rate for $t \leq \tau + w$, w is the prediction window. Also there is an assumption in Lin's online algorithm that the switching time is one time-slot. So the number of active servers at the time-slot τ using LCP(w), $x_{\tau}^{LCP(w)}$, should be decided at the beginning of the time-slot $\tau - 1$. Then the algorithm implies that at the beginning of the slot $\tau - 1$, the arriving rates $\lambda_1, \lambda_2, \ldots \lambda_{\tau}$ are known for $w = 0$.

In our simulation we set the prediction window used in Lin's algorithm $w = 0$, and we assume that at any time the arrival rates of the following two time-slots are known, these arrival rates are used in our algorithm to calculate the estimated future workload too.

In this simulation, the total energy using Lin's algorithm LCP(0) is 8100, the total energy using our online heuristic algorithm is 4690. The number of lost jobs using Lin's algorithm is 331, the number of lost jobs using our online heuristic is 132. The energy using our algorithm saves 42.1% energy compared to Lin's algorithm, and it also decrease the number of lost jobs by 60.1% compared with Lin's algorithm.

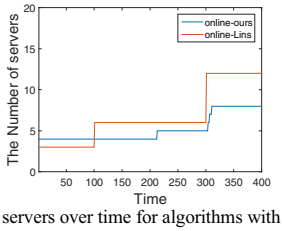

Fig.11 The number of servers over time for algorithms with a simple workload trace [1 2 2 4].

We test the impact of the short time-slot's duration as well. The small time-slot's duration determines the decision frequency of the servers for our online algorithm. Below are the graphs showing the number of servers over time for the two algorithms with different durations of short time-slots for our algorithm. The performance of our algorithm almost remains

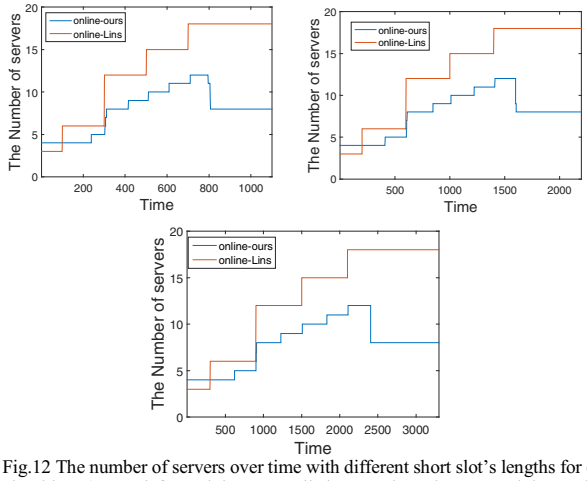

Fig.12 The number of servers over time with different short slot's lengths for our algorithm. (Upper left graph is 100 small slots per time-slot, upper right graph is 200 small slots per time-slot, the lower graph is 300 small slots per time-slot.).

Decision frequency of our algorithm	Energy (our algorithm)	the Number of Lost Jobs (our algorithm)	Energy (Lin's algorithm)	the Number of Lost Jobs (Lin's algorithm)
100 slots	14277	390	23100	1037
200 slots	14029	369	23100	1037
300 slots	14159	342	23100	1037

TABLE II. RESULTS WITH DIFFERENT SMALL TIME-SLOT'S LENGTH

the same with different slot durations. (Simulated with the workload trace [1 2 2 4 3 5 4 6 2 3 4].)

The comparison results demonstrate significant improvement of our scheme, even for very random workload. The main reason for the result is our model makes use of job's lifetime to arrange the servers to get smooth and relatively low workload, and also our algorithm allows jobs transfer between different servers.

VI. Related work

The topic of energy saving problem in data centers has been studied by a number of researchers. Some researchers develop algorithms based on Lyapunov Optimization technique ([1], [3], [8]), while considering different optimization modeling. Reference [1] focuses on reducing power cost along with queue stability and considers cost vs. delay trade-off; thus the approach taken by this work is suited for delay tolerant workloads, such as massively parallel and data intensive MapReduce jobs. Other researchers focus on virtualized data centers (e.g., [3], [10], [11]). For example, the reference [3] designed an algorithm that makes use of the queueing information available in the system to implicitly learn and adapt to unpredictable changes in the workload of virtualized data centers.

A different approach taken by [7] explored dynamic energy pricing and designed algorithm for achieving optimal geographical load balancing: routing to a data center further from the request source uses cheaper energy. The reference [10] considers collaborative filtering techniques, which are frequently used in recommendation systems; instead of learning each new workload in detail, the system leverages information it already has about applications it has seen to express the new workload as a combination of known applications. The Reference [5] suggests a simple last-empty-server-first job-dispatching strategy and each server independently solving a classic ski-rental problem.

We observed that in previous works the most common metric used in the optimization problem is job delay (e.g., [1], [8]), while the other aspects of QoS and the demand difference between different jobs haven't been fully studied. This is one of several aspects where our work differs – our algorithms incorporate latency tolerance measures (a lifetime) for each arriving job.

VII. Summary and Conclusions

In this paper, we have presented two novel heuristic algorithms for solving the right-sizing problem of data centers. As the problem of finding the optimal assignment is NP-complete, heuristics are necessary to provide practical solution. One particular aspect of our work that sets it apart from other works is that our algorithms incorporate latency tolerance measures (a lifetime) for each arriving job.

We first developed an offline heuristic, which is based on the observation that reducing the number of times that the servers are off and on has a benefit effect on the total energy consumption. We used this fact to "smooth" the servers' demand curve through leveraging the fact that jobs have some leeway when they are executed within their lifetime in the system. Based on the offline heuristic, we developed an online algorithm by predicting the future arrivals of jobs and their corresponding parameters.

We compared the performance of the two heuristics, the non-scheduling execution of the jobs, and the optimal algorithm. Our comparison results demonstrate that: (1) Our offline algorithm performs close to our online algorithm, (2) Both our heuristics perform close to the optimal solution, and (3) Both our algorithm reduce the total cost by more than 50%, compared to the non-scheduled case.

We also compared the performance of our heuristics with another scheme proposed in [12], which is referred to as the "Lin's algorithm". The results showed significant performance improvement of our algorithms.

Due to its low complexity and ability to handle large real-life scenarios, our algorithm presents a practical solution to data-centers right-sizing problem.

VIII. Acknowledgement

The work has been supported in part by the NSF grant numbers: CNS-1040689, ECCS-1308208 and CNS-1352880.

References

[1] Yao, Y., Huang, L., Sharma, A., Golubchik, L., & Neely, M. (2012, March). Data centers power reduction: A two time scale approach for delay tolerant workloads. In *INFOCOM, 2012 Proceedings IEEE* (pp. 1431-1439). IEEE.

[2] Lin, M., Wierman, A., Andrew, L. L., & Thereska, E. (2013). Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking (TON)*, *21*(5), 1378-1391.

[3] Urgaonkar, R., Kozat, U. C., Igarashi, K., & Neely, M. J. (2010, April). Dynamic resource allocation and power management in virtualized data centers. In *Network Operations and Management Symposium (NOMS), 2010 IEEE* (pp. 479-486). IEEE.

[4] Krioukov, A., Mohan, P., Alspaugh, S., Keys, L., Culler, D., & Katz, R. (2011). Napsac: Design and implementation of a power-proportional web cluster. *ACM SIGCOMM computer communication review*, *41*(1), 102-108.

[5] Lu, T., Chen, M., & Andrew, L. L. (2013). Simple and effective dynamic provisioning for power-proportional data centers. *Parallel and Distributed Systems, IEEE Transactions on*, *24*(6), 1161-1171.

[6] Guenter, B., Jain, N., & Williams, C. (2011, April). Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *INFOCOM, 2011 Proceedings IEEE* (pp. 1332-1340). IEEE.

[7] Liu, Z., Lin, M., Wierman, A., Low, S. H., & Andrew, L. L. (2011, June). Greening geographical load balancing. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems* (pp. 233-244). ACM.

[8] Ren, S., He, Y., & Xu, F. (2012, June). Provably-efficient job scheduling for energy and fairness in geographically distributed data centers. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on* (pp. 22-31). IEEE.

[9] Delimitrou, C., & Kozyrakis, C. (2013). Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGARCH Computer Architecture News*, *41*(1), 77-88.

[10] Beloglazov, A., Buyya, R., Lee, Y. C., & Zomaya, A. (2011). A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in computers*, *82*(2), 47-111.

[11] Kusic, D., Kephart, J. O., Hanson, J. E., Kandasamy, N., & Jiang, G. (2009). Power and performance management of virtualized computing environments via lookahead control. *Cluster computing*, *12*(1), 1-15.

[12] Lin, M., Wierman, A., Andrew, L. L., & Thereska, E. (2013). Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking (TON)*, *21*(5), 1378-1391.

[13] Adnan, M. A., Sugihara, R., Ma, Y., & Gupta, R. K. (2013, June). Energy-optimized dynamic deferral of workload for capacity provisioning in data centers. In *Green Computing Conference (IGCC), 2013 International* (pp. 1-10). IEEE.