Fast and energy-aware resource provisioning and task scheduling for cloud systems

Conference Paper · March 2017

DOI: 10.1109/ISQED.2017.7918312

6 authors, including:

Ji Li
University of Southern California
37 PUBLICATIONS **503** CITATIONS
SEE PROFILE

Shahin Nazarian
University of Southern California
71 PUBLICATIONS **862** CITATIONS
SEE PROFILE

Xue Lin
122 PUBLICATIONS **1,340** CITATIONS
SEE PROFILE

Yetang Wang
920 PUBLICATIONS **12,061** CITATIONS
SEE PROFILE

Fast and Energy-Aware Resource Provisioning and Task Scheduling for Cloud Systems

Hongjia Li[1], Ji Li[2], Wang Yao[3], Shahin Nazarian[2], Xue Lin[4], and Yanzhi Wang[1]
[1]Dept. of Electrical Engineering & Computer Science, Syracuse University, Syracuse, NY, USA
[2]Dept. of Electrical Engineering, University of Southern California, Los Angeles, CA, USA
[3]Middle School Department Shenzhen Experimental School, Shenzhen, China
[4]Dept. of Electrical & Computer Engineering, Northeastern University, Boston, MA, USA
[1]{hli42,ywang393}@syr.edu,[2]{jli724,shahin}@usc.edu,[3]YaoWangApplication@outlook.com,[4]xuelin@coe.neu.edu

***Abstract*—Cloud computing has become an attractive computing paradigm in recent years to offer on demand computing resources for users worldwide. Through Virtual Machine (VM) technologies, the cloud service providers (CSPs) can provide users the infrastructure, platform, and software with a quite low cost. With the drastically growing number of data centers, the energy efficiency has drawn a global attention as CSPs are faced with the high energy cost of data centers. Many previous works have contributed to improving the energy efficiency in data centers. However, the computational complexity may lead to unacceptable run time. In this paper, we propose a fast and energy-aware resource provisioning and task scheduling algorithm to achieve low energy cost with reduced computational complexity for CSPs. In our iterative algorithm, we divide the provisioning and scheduling to multiple steps which can effectively reduce the complexity and minimize the run time while achieving a reasonable energy cost. Experimental results demonstrate that compared to the baseline algorithm, the proposed algorithm can achieve up to 79.94% runtime improvement with an acceptable energy cost increase.**

***Index Terms*—Cloud computing, task scheduling, resource provisioning.**

I. Introduction

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storages, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. With Virtual Machine (VM) technologies, users can access the infrastructure, platform, and software without limitation of location, which significantly minimizes the operation cost for cloud service providers (CSPs). CSPs that own large data centers and server clusters can benefit through charging users from service access while users can implement these services on demand at competitive costs. In recent years, many computing service companies including Google, Yahoo, Microsoft and Amazon have rapidly established large scale platform to provide consumers cloud computing services. According to the estimation from Gartner, the worldwide public cloud service market will grow 16.5% in 2016 to a total of $204 billion [2].

Despite the high efficiency and convenience brought by these technologies, the growth in number and size of data centers inevitably leads to high energy costs. According to the latest United States Data Center Energy Usage Report released by the U.S. Department of Energy's Lawrence Berkeley National Laboratory [3], in 2014, data centers in the U.S. consumed an estimate of 70 billion kWh, equivalent to the electricity needed to power all the households in the states of Virginia and Washington combined [4]. And based on current trend estimates, U.S. data centers are projected to consume approximately 73 billion kWh in 2020. Hence, technical actions for minimizing energy consumption (energy cost) must be implemented including: (i) powering down unused and obsolete servers, and (ii) balanced provisioning of servers' resource utilization.

For their submitted workloads, the users have requirements on some basic service metrics such as response time, privacy, and security. In order to specify these metrics, the service level agreement (SLA) is in the place to provide a standardized service contract between CSPs and users. During the multiple-task operation, the CSP needs to satisfy the requirement of each task (specified in terms of SLA) and in the meantime aims to reduce the energy consumption (or energy cost) of the data center.

In this paper, we consider a general cloud computing paradigm with heterogeneous servers, executing a set of dependent tasks, modeled using task graph workload model [5], [6], [7], [8]. Figure 1 illustrates the overview of the cloud environment. Workloads are modeled as a large collection of directed graphs of tasks with output dependencies. Through the admission control policy, only the achievable tasks are selected by the CSP, while the others are rejected. Then according to user workload requests, the CSP starts to configure the appropriate amount of VMs and finally schedules the remaining tasks to meet the requirement of SLA while minimizing the energy consumption. For VM allocations, a pay-per-use economic model is adopted which can avoid wasting resources since charges are based on the actually utilized services, instead of provisioning for a certain amount of resources that may not be used.

Many prior works have focused on improving the energy efficiency while satisfying the SLA under this cloud computing framework. The authors in [9] propose a polynomial-time heuristic method to transform the problem into a probability-based load balancing problem that is then solved with worst-fit decreasing bin-packing heuristic. However they assume system homogeneity as well as homogeneous and independent tasks. In order to consider the heterogeneous resources, the authors in [10] propose a genetic algorithm for task/VM scheduling

and energy optimization in a holistic manner. However, the genetic algorithm has scalability problems. In order to achieve a flexible and scalable scheduling algorithm with energy cost savings, the authors in [5] develop a negotiation-based cloud resource provisioning and task scheduling algorithm. The proposed algorithm adopts an iterative framework and a congestion model, , where the negotiation-based algorithm has been previously applied to the Smart Grid scheduling problems [11]. However, the algorithm results in high complexity, which consequently increases the computation time.

In this paper, we propose a novel algorithm that solves the provisioning and scheduling problem step by step to achieve low complexity while maintaining the quality of solution in terms of energy cost. Inspired by [5], the iterative manner is introduced. In each iteration, the proposed algorithm first configures appropriate type of VMs for each task, then places those VMs on selective servers based on a greedy algorithm, and finally schedules all the tasks with the negotiation technology to achieve the minimum energy consumption, without violating the SLA. Experimental results demonstrate that the proposed algorithm can achieve up to 79.94% runtime improvement with acceptable energy cost increase as compared to the baseline algorithm.

The rest of this paper is organized as follows. Section II shows the system model. Section III presents the proposed VM allocation, resource provisioning and task scheduling algorithm. Section IV reports the experimental results, and the paper is concluded in Section V.

II. System Model

A. Workload Model

The workload is modeled as a collection of disjoint directed acyclic graphs (DAGs), and the whole workload consists of M user workload requests: $\{G_1(N_1, E_1), G_2(N_2, E_2), ..., G_M(N_M, E_M)\}$ which comes from separate users. In each DAG G_a $(1 \leq a \leq M)$, vertex T_i^a $(1 \leq i \leq N_a)$ represents a single task. The total number

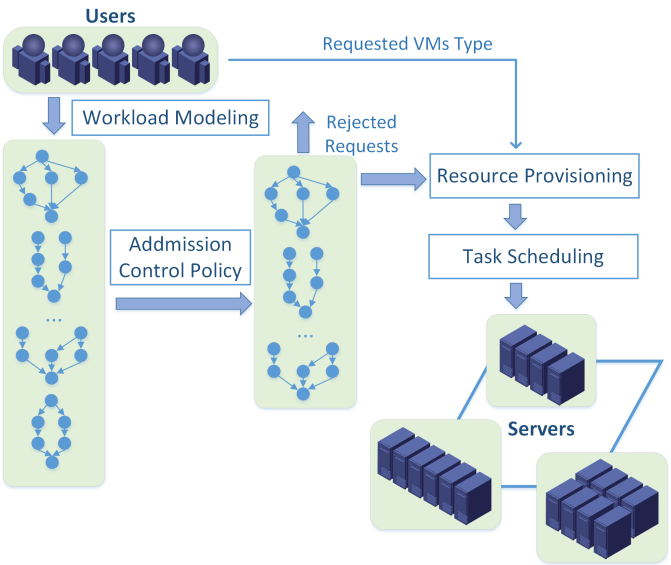

Fig. 1: The cloud environment overview.

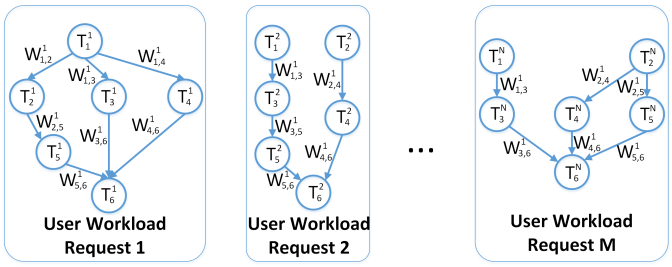

Fig. 2: An example of workload requests.

of tasks is denoted by N_{total}, $N_{total} = \Sigma_{a=1}^{M} N_a$. Each edge $E_a(i,j)$ from task (T_i^a) to task (T_j^a) in G_a indicates that task T_j^a depends on the output of task T_i^a, and the weight $W_{i,j}^a$ of the edge represents the amount of data that needs to be delivered from the precedent task (T_i^a) to the following task (T_j^a). Figure 2 presents an example of M user workload requests.

VMs supported by the CSP can be classified into L different types: $\{VM_1, VM_2, ..., VM_L\}$. Each type VM_g $(1 \leq g \leq L)$ is associated with a two-tuple parameter set $\{R_{CPU}^g, R_{MEM}^g\}$, where R_{CPU}^g and R_{MEM}^g represent the required amount of CPU and memory resource by VM_g, respectively. Similarly, a two-tuple set $\{D_{CPU}^{a,i}, D_{MEM}^{a,i}\}$ is associated with each task T_i^a indicating the required amount of CPU and memory resource to run task T_i^a. Hence, one prerequisite of successful task execution is sufficient resource, i.e., $D_{CPU}^{a,i} \leq R_{CPU}^g$ and $D_{MEM}^{a,i} \leq R_{MEM}^g$, if T_i^a is allocated with VM_g.

Additionally, each task comes together with another set $\{K_i^a, L_i^a(g)\}$, in which K_i^a and $L_i^a(g)$ denote the types of VMs that can accommodate task T_i^a and the estimated execution time associated with VM type g, respectively. The aforementioned information is given as input data.

Each user workload request specifies a deadline $T_{deadline}^a$, and the scheduled start time of task T_i^a is denoted by s_i^a. According to the SLA, each task T_i^a in the user workload request must be completed before the deadline $T_{deadline}^a$, i.e., $s_i^a + L_i^a(g) \leq T_{deadline}^a, \forall T_i^a$. To meet the deadline requirement in SLA, the CSP can allocate fast VMs or schedule the user request early. However, when it is impossible to finish a user request before the deadline even with the fastest VM allocated and earliest start time scheduled, admission control should be performed to reject such a request.

B. Cloud Platform Model

The CSP provides a group of P servers: $\{S_1, S_2, ..., S_P\}$ in which several close servers can form a server farm with local connections. Among different farms, data can be transferred through high speed channels. We model the cloud platform with P servers as an undirected graph. Each node S_x denotes an individual server, and the weight $R_{x,y}$ of each edge (S_x, S_y), embodies the communication capacity between server S_x and S_y $(1 \leq x, y \leq P)$. $R_{x,y}$ reflects the distance between servers and channel bandwidth. Note that $R_{x,x} = \infty, \forall x \in [1, P]$, i.e. data transfer on the same sever does not lead to any communication overhead. Figure 3 presents an example of a cloud platform consisting of three server farms.

Each server S_x is coupled with a two-tuple parameter set $\{C^x_{CPU}, C^x_{MEM}\}$. C^x_{CPU} and C^x_{MEM} denote the total amount of CPU and memory resource available on server S_x, respectively. During the operation, an integer array $\mathbf{Q^x}$ consisting of L elements: $\{Q^x_1, Q^x_2, ..., Q^x_L\}$ is used to represent the state of server S_x, where Q^x_g indicates the number of type g VMs that are hosted on S_x. Since the allocation may change over time, we present the $\mathbf{Q^x}$ at time t as $\mathbf{Q^x(t)}$. One obvious prerequisite for VM allocation is that the total required amount of resource for all the VMs allocated in each server must not be larger than the available resource in that server during the entire operation time: $\forall x$ and $\forall t$, $\Sigma^L_{g=1} Q^x_g(t) \cdot R^g_{CPU} \leq C^x_{CPU}$ and $\Sigma^L_{g=1} Q^x_g(t) \cdot R^g_{MEM} \leq C^x_{MEM}$.

C. Energy Consumption Model and Price Model

The total power consumption of each server S_x at time t is composed of static power consumption $P^x_{static}(t)$ and dynamic power consumption $P^x_{dynamic}(t)$. Both items are associated with the utilization rate $U_x(t)$, which denotes the ratio of CPU usage at time t on S_x. The utilization rate $U_x(t)$ is calculated as

$$U_x(t) = \frac{\Sigma^L_{g=1} Q^x_g(t) \cdot R^g_{CPU}}{C^x_{CPU}} \times 100\% \quad (1)$$

$P^x_{static}(t)$ is constant when $U_x(t) > 0$, and 0 otherwise. On the other hand, $P^x_{dynamic}(t)$ has a more complex relationship with $U_x(t)$. For each server S_x, the optimal utilization rate $Uopt_x$ affects the power consumption significantly. When $U_x(t) > Uopt_x$, the dynamic power consumption has a more radical increase rate than the other side. Generally, $Uopt_x$ for modern servers is around 0.7 [12]. The dynamic power consumption model in [10] is adopted, where $P^x_{dynamic}(t)$ is calculated as

$$\begin{cases} \mu_x \cdot U_x(t), & \text{when } U_x(t) < Uopt_x \\ \mu_x \cdot Uopt_x + \gamma_x \cdot \left(U_x(t) - Uopt_x\right)^2, & \text{otherwise} \end{cases} \quad (2)$$

where μ_x and γ_x represent the power consumption increase coefficients.

In this paper, we consider a combined price model $\zeta\left(t, P^x_{static}(t) + P^x_{dynamic}(t)\right)$, which includes a time-of-use (TOU) price component and a power consumption-dependent price component, under a general dynamic pricing policy [13], [14], [15]. More specifically, the TOU price is much cheaper in off-peak period so as to incentivize users to avoid peak usage period, whereas the power consumption-dependent price is constantly increasing with respect to the instantaneous power consumption, in order to penalize peak energy consumption.

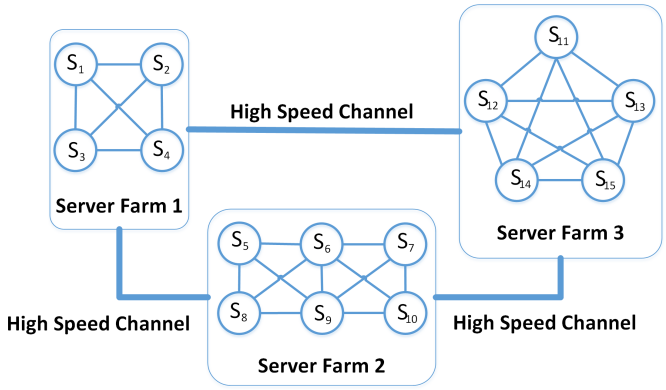

Fig. 3: The example server farms.

The entire scheduling period is determined by the largest schedule length among all workload requests which is denoted by T_{max}. The total power consumption is the sum of the power consumption through all the servers for the entire operation time, and the total energy cost is the total electric cost for the CSP.

D. Problem Statement

Based on the models described above, the statement of VM allocation, resource provisioning and task scheduling problem is as follows:

VM Allocation, Resource Provisioning and Task Scheduling (ARPTS) Problem.

Given user workloads and cloud platform information.

Find the VM allocation, the server provisioning and start time for each task.

Minimize:

$$Total\ Cost = \sum_{t=1}^{T_{max}} \zeta\left(t, \sum_{x=1}^{P}\left(P^x_{static}(t) + P^x_{dynamic}(t)\right)\right) \quad (3)$$

Subject to:

$$\Sigma^L_{g=1} Q^x_g(t) \cdot R^g_{CPU} \leq C^x_{CPU},\ \forall t \text{ and } \forall x \in [1, P] \quad (4)$$

$$\Sigma^L_{g=1} Q^x_g(t) \cdot R^g_{MEM} \leq C^x_{MEM},\ \forall t \text{ and } \forall x \in [1, P] \quad (5)$$

$$D^{a,i}_{CPU} \leq R^g_{CPU}, \forall T^a_i \quad (6)$$

$$D^{a,i}_{MEM} \leq R^g_{MEM}, \forall T^a_i \quad (7)$$

$$L^a_i(g) + s^a_i \leq T^a_{deadline}, \forall T^a_i \quad (8)$$

and *the task dependency requirements.*

The input of the ARPTS problem has a high dimension (i.e., user workloads, time period, constraints, servers, etc.). Finding the optimal solution in such a problem with a large set of parameters using nonlinear optimizers is impractical because of the unacceptable runtime [16]. Therefore, instead of finding the optimal solution, we seek for a sufficiently good solution, and an essential part of the proposed algorithm is to improve the computational complexity through reduction of the problem dimension while preserving the solution quality using efficient heuristics.

III. Fast and Energy-Aware Resource Provisioning and Task Scheduling Algorithm

A. Motivation

Cloud cost management provides a significant opportunity for savings, and actions for optimizing cloud costs include shutting down unused servers, scheduling tasks in off-peak periods, and selecting low-power servers [17]. Existing works [10], [5] have focused on the improvement of solution quality. However, with the massive growth in the scale of data processed in the cloud platform and the fast growing numbers of both users and servers, the runtime improvement of the

resource provisioning and task scheduling algorithm is becoming important. Without careful optimization, the runtime of the algorithm may be unacceptable for a large-scale cloud platform. Therefore, unlike prior works that have concentrated on the solution quality, this work mainly focuses on the improvement of the computational complexity.

To be more specific, in the ARPTS problem described in Section II-D, the search space of configuring and scheduling one task in a user workload request has a high dimension, i.e., VMs, servers, and time period. One important observation is that the search can be divided into three separate steps: (i) VM allocation, (ii) resource provisioning, and (iii) task scheduling. Since decomposing these steps can improve the computational complexity to $O(step1+step2+step3)$ instead of $O(step1 \times step2 \times step3)$, effective heuristics should be applied to divide the decision making process into separate steps.

B. Fast and Energy-Aware Resource Provisioning and Task Scheduling Algorithm

In this section, we present the proposed Fast and Energy-Aware Resource Provisioning and Task Scheduling (FERPTS) algorithm for solving the ARPTS problem.

As mentioned in Section-I, an iterative manner is adopted for the entire algorithm. The algorithm terminates when the total cost in Eqn. (3) is not decreased for h_{set} consecutive iterations, or when it reaches the maximum iteration number h_{max}. More specifically, in each iteration, all the decisions from the previous iteration are ripped-up (i.e., cleaned), and then all tasks will be re-decided one by one, including the VM allocation, resource provisioning and scheduling. In addition, at the beginning of each iteration, the ready tasks can be selected through the admission control policy. Clearly, the complexity of the entire algorithm is equal to h_{max} times the complexity of processing one iteration. As a result, improving the complexity of the algorithm is equivalent to improving the complexity of processing one iteration in the algorithm.

In this paper, we decompose the decision making process in one iteration into three steps, (i) VM allocation, (ii) resource provisioning, and (iii) task scheduling, as follows:

Step 1: VM Allocation

The purpose for this step is to allocate the most satisfactory VM for task T_i^a concerning the CPU and memory resource. During the operation, each task is mapped to one VM and we assume that all tasks are mapped to a single server with unlimited CPU and memory resources. It is observed that allocating a task into a VM with significantly larger amount of CPU/memory resource than the requested will lead to unnecessary high utilization and high power consumption, because the utilization rate of the server is calculated based on the CPU occupied by the allocated VMs in that server, not the requested CPU resources from the tasks (according to Eqn. (1)). In order to maximize the utilization of VM and avoid inefficiency during VM allocation, we allocate the VM with minimal gap between the available CPU and memory resources in the VM and the requested amount of CPU and memory of the task T_i^a. In this way, the computational complexity can be effectively reduced by selecting only one VM for each task instead of trying all VM options in a brute force manner. The complexity of step 1 is $O(N_{total} \cdot L)$.

Step 2: Resource Provisioning

After the VM is configured for the task T_i^a, the next step is to provision the server. It is difficult to completely decompose the server selection (i.e., resource provisioning) step and the task scheduling step. Instead, we conduct the resource provisioning step with a simplified task scheduling algorithm, i.e., the greedy algorithm, and then in the following task scheduling step, we refine the scheduling results using a negotiation-based task scheduling algorithm that improves the solution quality, while maintain the same level of computational complexity.

We denote the resource provisioning decision in the h-th iteration after scheduling the r-th task ($1 \leq r \leq N_{total}$) by Z_r^h. Since the power consumption of server S_x depends on the provisioning decision, we denote $P_{static}^x(t)$ and $P_{dynamic}^x(t)$ after the decision Z_r^h as $P_{static}^x(t, Z_r^h)$ and $P_{dynamic}^x(t, Z_r^h)$, respectively. The *Cost Increase* (CI) in the h-th iteration after scheduling the r-th task in the t-th time slot is calculated by

$$CI = \sum_{t=1}^{T_{max}} \left(\zeta\left(t, \sum_{x=1}^{P} \left(P_{static}^x(t, Z_r^h) + P_{dynamic}^x(t, Z_r^h)\right)\right) - \zeta\left(t, \sum_{x=1}^{P} \left(P_{static}^x(t, Z_{r-1}^h) + P_{dynamic}^x(t, Z_{r-1}^h)\right)\right)\right) \tag{9}$$

For the server resource provisioning, we choose the Z_r^h with the minimum CI for each task. The complexity of this resource provisioning step is $O(N_{total}^2 \cdot P \cdot T_{deadline}^{max})$, where $T_{deadline}^{max}$ is the largest deadline among all tasks. The N_{total}^2 term is caused by finding each ready task and for each ready task, finding the server and scheduling.

Step 3: Task Scheduling

Inspired by [5], we adopt the negotiation-based technology to refine the schedule of each task after the VM allocation and server provisioning.

In each iteration, it is observed that selecting the time slot that has been already occupied by other tasks can lead to a high congestion, i.e., high cost in that time slot. To avoid such congestion, an intra-iteration congestion term $C(t)$ is introduced, which denotes the total number of tasks that has been scheduled to time slot t within an iteration. The scheduler will be tempted to explore other scheduling possibilities since increased $C(t)$ causes a high cost penalty, cf. Eqn. (10). In addition, another inter-iteration term $B_a(t)$ is introduced which indicates the total number of times a user workload request a violates the deadline requirement in Eqn. (8) at time slot t in the previous iterations. The deadline violation will gradually lead to a high energy cost as $B_a(t)$ increases, which can incentivize the scheduler to schedule this workload request to other time slots so as to satisfy the deadline requirement. With the two terms mentioned above, the CI in the h-th

iteration after the r-th task is modified as:

$$CI' = \sum_{t=1}^{T_{max}} \left(\zeta\left(t, \sum_{x=1}^{P} \left(P^x_{static}(t, Z^h_r) + P^x_{dynamic}(t, Z^h_r)\right)\right) - \zeta\left(t, \sum_{x=1}^{P} \left(P^x_{static}(t, Z^h_{r-1}) + P^x_{dynamic}(t, Z^h_{r-1})\right)\right)\right) \cdot \left(1 + \alpha \cdot C(t)\right) + \sum_{t=1}^{T_{max}} \beta \cdot B_a(t) \tag{10}$$

where α and β denote the weight of $C(t)$ and $B_a(t)$, respectively. Term $C(t)$ and $B_a(t)$ are initially set to zeros, in order to make the initial cost increase the same as Eqn. (9). Note that the intra-iteration congestion term $C(t)$ is updated after each workload request is scheduled within one iteration, whereas the inter-iteration term $B_a(t)$ is updated at the end of each iteration to transfer the information to the future iterations. The complexity of this step is $O(N^2_{total} \cdot T^{max}_{deadline})$.

The total complexity of proposed FERPTS algorithm is $O(N_{total} \cdot L + N^2_{total} \cdot P \cdot T^{max}_{deadline} + N^2_{total} \cdot T^{max}_{deadline})$, which is equal to $O(N^2_{total} \cdot P \cdot T^{max}_{deadline})$ since the VM number L is significant smaller than the time slot number $T^{max}_{deadline}$. Algorithm 1 illustrates the pseudo code of the proposed algorithm.

Algorithm 1: Fast and Energy-Aware Resource Provisioning and Task Scheduling Algorithm

```
Initialize user workload model, cloud platform model, energy consumption model and terms;
Set iteration counter h = 0;
repeat
    h = h + 1;
    foreach time t do C(t) = 0;
    foreach task do
        re-decide all provisioning and scheduling ;
    end
    repeat
        Find a ready task T_i^a;
        /* step1: VM allocation                          */
        Allocate the VM type for the task T_i^a with the minimum resource gap ;
        /* step2: resource provisioning                  */
        Configure the servers for the task T_i^a with the minimum CI in Eqn. (9) ;
        /* step3: task scheduling                        */
        Schedule the task T_i^a such that CI' in Eqn. (10) is minimized;
        Update the intra-iteration term C(t);
    until all tasks are configured and scheduled;
    foreach job a do
        foreach time slot t do
            Update inter-iteration term B_a(t);
        end
    end
    Calculate total energy cost in Eqn. (3);
until total cost not decreased for h_set iterations or h > h_max;
return configuration and scheduling of each task;
```

IV. Experimental Results

In this section, we compare our proposed algorithm with the negotiation-based NBRPTS algorithm [5] and a greedy baseline which schedules all the ready tasks with the best start time according to the cost increase in Eqn. (9). We assume the cloud platform includes up to 5 server farms and 30 heterogeneous servers. With the information mentioned in Section II-B, the speed of data transfer in local connections within a server farm is faster than in high speed channel among different server farms. We randomly generate user workload request model information including the number of tasks in each user workload request and the dependencies between tasks. We extract the CPU and memory resource requirements of each task from Google cluster data [18], which represents 29 day's worth of cell information from May 2011, on a cluster of about 12.5k machines. In addition, the amount of resources in VMs and servers are generated randomly. The summary of the key parameters in our models are illustrated in Table I.

We first compare the performance including run time and energy cost through the Google 29-day period with a fixed cloud platform that consists of 4 server farms and 30 servers, and 20 user workload requests. Figure 4 (a) and Figure 5 (a) show the comparison results during the 29 days. In order to see the effects of workload requests and servers, we next use the resource information from the first day to sweep the number of user workload requests from 5 to 20 with a fixed cloud platform (with results shown in Figure 4 (b) and Figure 5 (b)) and then change the server number from 10 to 30 with the same user workload requests (with results shown in Figure 4 (c) and Figure 5 (c)). We analyze the performance from two aspects: (i) run time, and (ii) energy cost, as follows:

TABLE I: The summary of modeling parameters

Cloud Platform Parameters		User Workload Characteristics		
Servers Farms	Servers	Total User Workload Requests	Task per Request	Task Latency
2-5	10-30	5-20	5-14	2-8

A. Run Time

Our main objective is to reduce the run time while maintaining the low energy cost. We compare our proposed FERPTS algorithm with previous NBRPTS algorithm and the greedy baseline under different number of user workload requests and servers. As shown in Figure 4, the run time is significantly reduced using FERPTS algorithm. During the 29-day period, the average run time of the proposed algorithm is 61.65s, while the average run time of NBRPST and baseline are 307.30s and 290.44s, respectively. The proposed algorithm achieves up to 79.94% run time savings compared to baseline.

B. Energy Cost

For the energy cost comparisons, we use the normalized energy cost which is the ratio between the energy cost of the proposed algorithm and the energy cost of the baselines. In Figure 5 we show the normalized energy cost of these three algorithms under different conditions. Compared with the baseline algorithm and the NBRPTS algorithm, the averaged increase of energy cost caused by the proposed algorithm is 11.57% and 11.26%, respectively. Considering the significant run time improvement, this amount energy cost increase can be acceptable.

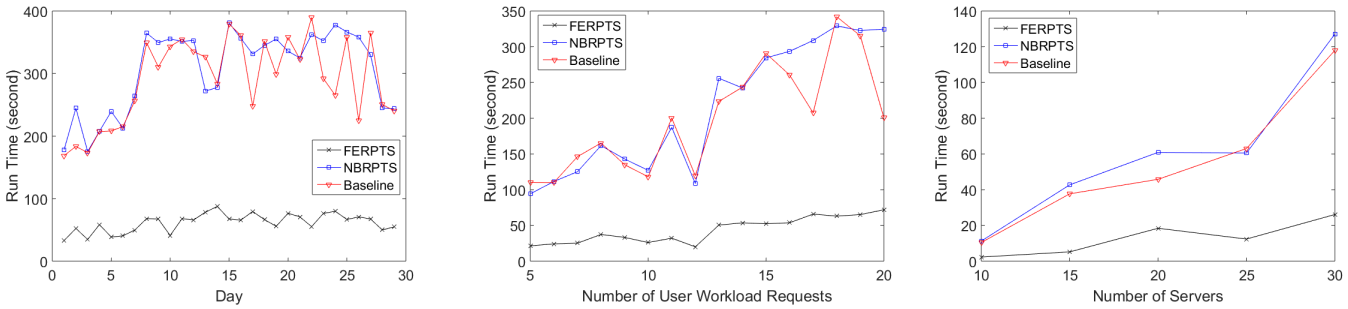

Fig. 4: The run time comparisons of three algorithm (FERPTS, NBRPTS and baseline). (a) Comparison over the 29-day period. (b) Comparison over the user workload requests number. (c) Comparison over the servers number.

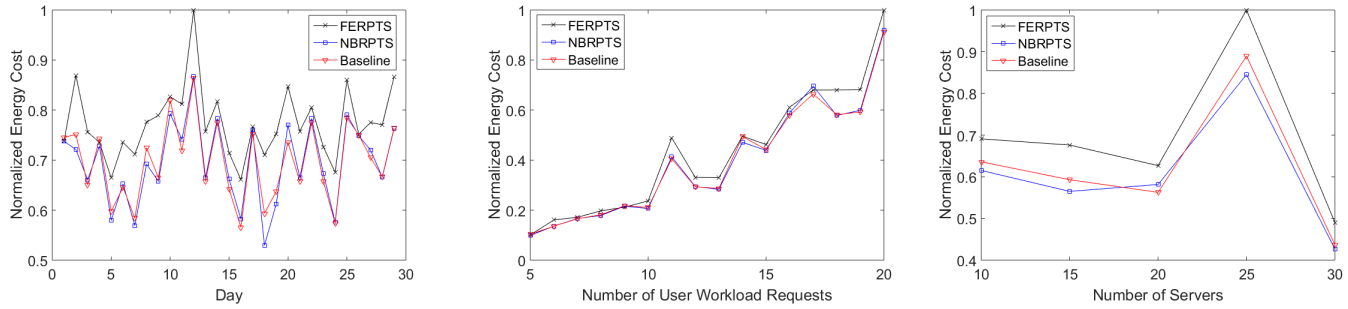

Fig. 5: The energy cost comparisons of three algorithm (FERPTS, NBRPTS and baseline). (a) Comparison over the 29-day period. (b) Comparison over the user workload requests number. (c) Comparison over the servers number.

V. Conclusion

In this paper, we propose an algorithm to achieve the minimum run time with a low energy cost. In order to effectively reduce the total computational complexity of the algorithm, we decompose the decision making process into three steps: VM allocation, resource provisioning and task scheduling. The experimental results show that the proposed algorithm can achieve up to 79.94% run time savings compared to the baseline while with an acceptable energy cost increase.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[2] "Gartner forecast 2016." [Online]. Available: http://www.gartner.com/newsroom/id/3188817

[3] A. Shehabi, S. J. Smith, D. A. Sartor, R. E. Brown, M. Herrlin, J. G. Koomey, E. R. Masanet, N. Horner, I. L. Azevedo, and W. Lintner, "United states data center energy usage report," 06/2016 2016.

[4] "The power of efficiency to cut data center energy waste." [Online]. Available: https://www.nrdc.org/experts/pierre-delforge/power-efficiency-cut-data-center-energy-waste

[5] J. Li, Y. Wang, X. Lin, S. Nazarian, and M. Pedram, "Negotiation-based resource provisioning and task scheduling algorithm for cloud systems," in *2016 17th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2016, pp. 338–343.

[6] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[7] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming*, vol. 14, no. 3-4, pp. 217–230, 2006.

[8] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[9] X. Zhong and C.-Z. Xu, "Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee," *IEEE Transactions on Computers*, vol. 56, no. 3, pp. 358–372, 2007.

[10] Y. Gao, Y. Wang, S. K. Gupta, and M. Pedram, "An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 2013, p. 31.

[11] J. Li, Y. Wang, X. Lin, S. Nazarian, and M. Pedram, "Negotiation-based task scheduling and storage control algorithm to minimize user's electric bills under dynamic prices," in *The 20th Asia and South Pacific Design Automation Conference*. IEEE, 2015, pp. 261–266.

[12] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[13] T. Cui, Y. Wang, H. Goudarzi, S. Nazarian, and M. Pedram, "Profit maximization for utility companies in an oligopolistic energy market with dynamic prices," in *Online Conference on Green Communications (GreenCom), 2012 IEEE*. IEEE, 2012, pp. 86–91.

[14] J. Li, Y. Wang, T. Cui, S. Nazarian, and M. Pedram, "Negotiation-based task scheduling to minimize user's electricity bills under dynamic energy prices," in *Green Communications (OnlineGreencomm), 2014 IEEE Online Conference on*. IEEE, 2014, pp. 1–6.

[15] T. Cui, S. Chen, Y. Wang, Q. Zhu, S. Nazarian, and M. Pedram, "Optimal co-scheduling of hvac control and battery management for energy-efficient buildings considering state-of-health degradation," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 775–780.

[16] R. Hemmecke, M. Köppe, J. Lee, and R. Weismantel, "Nonlinear integer programming," in *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 561–618.

[17] K. Weins, "Cloud computing trends: 2016 state of the cloud survey." [Online]. Available: http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2016-state-cloud-survey

[18] "Google cluster data." [Online]. Available: https://github.com/google/cluster-data