Real-Time Tasks Oriented Energy-Aware Scheduling in Virtualized Clouds

Xiaomin Zhu, *Member, IEEE*, Laurence T. Yang, *Member, IEEE*, Huangke Chen, Ji Wang, Shu Yin, *Member, IEEE*, and Xiaocheng Liu

Abstract—Energy conservation is a major concern in cloud computing systems because it can bring several important benefits such as reducing operating costs, increasing system reliability, and prompting environmental protection. Meanwhile, power-aware scheduling approach is a promising way to achieve that goal. At the same time, many real-time applications, e.g., signal processing, scientific computing have been deployed in clouds. Unfortunately, existing energy-aware scheduling algorithms developed for clouds are not real-time task oriented, thus lacking the ability of guaranteeing system schedulability. To address this issue, we first propose in this paper a novel rolling-horizon scheduling architecture for real-time task scheduling in virtualized clouds. Then a task-oriented energy consumption model is given and analyzed. Based on our scheduling architecture, we develop a novel energy-aware scheduling algorithm named EARH for real-time, aperiodic, independent tasks. The EARH employs a rolling-horizon optimization policy and can also be extended to integrate other energy-aware scheduling algorithms. Furthermore, we propose two strategies in terms of resource scaling up and scaling down to make a good trade-off between task's schedulability and energy conservation. Extensive simulation experiments injecting random synthetic tasks as well as tasks following the last version of the Google cloud tracelogs are conducted to validate the superiority of our EARH by comparing it with some baselines. The experimental results show that EARH significantly improves the scheduling quality of others and it is suitable for real-time task scheduling in virtualized clouds.

Index Terms—Virtualized cloud, real-time, energy-aware, scheduling, rolling-horizon, elasticity

1 Introduction

The cloud, consisting of a collection of interconnected and virtualized computers dynamically provisioned as one or more unified computing resource(s), has become a revolutionary paradigm by enabling on-demand provisioning of applications, platforms, or computing resources for customers based on a "pay-as-you-go" model [1]. Nowadays, an increasing number of enterprises and governments have deployed their applications including commercial business and scientific research in clouds motivated by the reasonable price as they are offered in economy of scale, and shifting responsibility of maintenance, backups, and license management to cloud providers [2]. Hence, some IT companies are significantly benefited from cloud providers by relieving them from the necessity in setting up basic hardware and software infrastructures, enabling more attention to the innovation and development for their main pursuit [3].

It is worthwhile to note that to provide cloud services, more and more large-scale data centers containing thousands of computing nodes are built, which results in consuming tremendous amount of energy with huge cost [5]. Moreover, high energy consumption causes low system reliability since the Arrhenius life-stress model shows that the failure rate of electronic devices will double as the temperature rises by every 10 °C [6]. Furthermore, high energy consumption has negative impacts on environment. It is estimated that computer usage accounts for 2 percent of anthropogenic CO_2 emission. Data center activities are estimated to release 62 million metric tons of CO_2 into the atmosphere [7]. Consequently, it is highly indispensable to employ some measures to reduce energy consumption of cloud data centers and make them energy efficient.

One of the important reasons about the extremely high energy consumption in cloud data centers can be attributed to the low utilization of computing resources that incurs a higher volume of energy consumption compared with efficient utilization of resources. The resources with a low utilization still consume an unacceptable amount of energy. According to recent studies, the average resource utilization (RU) in most of the data centers is lower than 30 percent [8], and the energy consumption of idle resources is more than 70 percent of peak energy [9]. In response to the poor resource utilization, virtualization technique is an efficient approach to increase resource utilization and in turn reduce energy consumption. The virtualization technique enables multiple virtual machines (VMs) to be placed on the same physical hosts and supports the live migration of VMs between physical hosts based on the performance requirements. When VMs do not use all the provided resources, they can be logically resized and consolidated to the minimum number of physical hosts, while idle nodes can be switched to sleep or hibernate mode to eliminate the idle

- *X. Zhu, H. Chen, J. Wang, and X. Liu are with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha, Hunan 410073, China. E-mail: {xmzhu, hkchen, wangji}@nudt.edu.cn, nudt200203012007xcl@gmail.com.*
- *L.T. Yang is with the Department of Computer Science, St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada. E-mail: ltyang@stfx.ca.*
- *S. Yin is with the School of Information Science and Engineering, Hunan University, Changsha, Hunan 410012, China. E-mail: shuyin@hnu.edu.cn.*

Manuscript received 18 Sept. 2013; revised 6 Jan. 2014; accepted 18 Feb. 2014. Date of publication 20 Apr. 2014; date of current version 30 July 2014.
Recommended for acceptance by I. Bojanova, R.C.H. Hua, O. Rana, and M. Parashar.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2310452

energy consumption and thus reduce the total energy consumption (TEC) in cloud data centers [3].

Noticeably, many applications, e.g., scientific computing, signal processing, deployed in clouds are with real-time nature, in which the correctness depends not only on the computation results, but also on the time instants at which these results are produced. For some applications, it is even mandatory in some cases to provide real-time guarantee. For example, if a photograph processing application is deployed in clouds, when an earthquake occurs, there is an urgent need for cloud data centers to process photographs of affected areas obtained by satellites in a real-time manner. Generally, the processed photographs are expected to be acquired within a few hours or even in dozens of minutes for timely conducting damage assessments and making rescue planing. Missing deadlines on getting photographs may greatly affect damage assessment and rescue, resulting in catastrophic consequence. Another example, taken from [4], is a real-time signal processing application that can be deployed in a cloud. In this application, the signal processing task should be finished within timing constraint, otherwise, the signal quality will be greatly degraded. It is obvious that guaranteeing real-time response for these kinds of applications is more crucial than energy conservation. Hence, more physical hosts in a cloud should be active to finish these real-time tasks before their deadlines. In contrast, if some real-time tasks are not emergent in nature, the cloud data center may use less physical hosts to reduce energy consumption while satisfying the timing requirements of users.

Motivation. It is well-known that scheduling is an efficient software technique to achieve high performance for applications running in clouds. Unfortunately, to the best of our knowledge, most existing energy-aware scheduling algorithms do not sufficiently consider real-time task allocation and energy saving simultaneously in clouds. Commonly, the workload is assumed in traditional energy-saving scheduling, and task allocation and resource provisioning are separated, which presents a big challenge for us to devise novel energy-aware scheduling strategies for real-time tasks in clouds so as to bridge the gap. To address this issue, we attempt to in this paper incorporate the cloud elasticity and tasks' real-time guarantee into energy-aware scheduling strategies. Specifically, our approach first gives high priority to deal with schedulability, i.e., if some real-time tasks cannot be finished before their deadlines using current active hosts, it will add new VMs or hosts to finish as many tasks as possible even though much energy consumption may be produced. When some hosts are sitting idle, our approach strives to reduce energy consumption by consolidating VMs while achieving high schedulability.

Contribution. The major contributions of this paper are as follows:

- We proposed an energy-aware scheduling scheme by rolling-horizon (RH) optimization, and we introduced an enabling scheduling architecture for rolling-horizon optimization.
- We developed some policies for VMs' creation, migration and cancellation to dynamically adjust the scale of cloud, meeting the real-time requirements and striving to save energy.
- We put forward an Energy-Aware Rolling-Horizon scheduling algorithm or EARH for real-time independent tasks in a cloud.

The rest of this paper is organized as follows: In Section 2, we summarize the related work in the literature. Section 3 gives the problem depictions. In Section 4, we describe the EARH algorithm and the main principles behind it are discussed. Section 5 presents the simulation experiments and performance analysis. Section 6 concludes this paper with a summary and future work.

2 Related Work

Green computing and energy conservation in modern distributed computing context are receiving a great deal of attention in the research community and efficient scheduling methods in this issue have been overwhelmingly investigated [19], [23], [24]. In a broad sense, scheduling algorithms can be classified into two categories: static scheduling and dynamic scheduling [10]. Static scheduling algorithms make scheduling decisions before tasks are submitted, and are often applied to schedule periodic tasks [11]. However, aperiodic tasks whose arrival times are not known a priori must be handled by dynamic scheduling algorithms (see, for example, [12], [13]). In this study, we focus on scheduling aperiodic and independent real-time tasks.

Chase et al. considered the energy-efficient management issue of homogeneous resources in Internet hosting centers. The proposed approach reduces energy consumption by switching idle servers to power saving modes and is suitable for power-efficient resource allocation at the data center level [14]. Zikos and Karatza proposed a performance and energy-aware scheduling algorithm in cluster environment for compute-intensive jobs with unknown service time [13]. Ge et al. studied distributed performance-directed DVFS scheduling strategies that can make significant energy savings without increasing execution time by varying scheduling granularity [15]. Kim et al. proposed two power-aware scheduling algorithms (space-shared and time-shared) for bag-of-tasks real-time applications on DVS-enabled clusters to minimize energy dissipation while meeting applications' deadlines [16]. Nélis et al. investigated the energy saving problem for sporadic constrained-deadline real-time tasks on a fixed number of processors. The proposed scheduling algorithm is preemptive; each process can start to execute on any processor and may migrate at runtime if it gets preempted by earlier-deadline processes [17]. It should be noted that these scheduling schemes do not consider resource virtualization, the most important feature of clouds, thus they cannot efficiently improve the resource utilization in clouds.

Nowadays, virtualization technology has become an essential tool to provide resource flexibly for each user and to isolate security and stability issues from other users [18]. Therefore, an increasing number of data centers employ the virtualization technology when managing resources. Correspondingly, many energy-efficient scheduling algorithms for virtualized clouds were designed. For example, Liu et al. aimed to reduce energy consumption in virtualized data centers by supporting virtual machine migration and

VM placement optimization while reducing the human intervention [19]. Petrucci et al. presented the use of virtualization for consolidation and proposed a dynamic configuration method that takes into account the cost of turning on or off servers to optimize energy management in virtualized server clusters [20]. Bi et al. suggested a dynamic resource provisioning technique for cluster-based virtualized multitier applications. In their approach, a hybrid queuing model was employed to determine the number of VMs at each tier [21]. Verma et al. formulated the power-aware dynamic placement of applications in virtualized heterogeneous systems as continuous optimization, i.e., at each time frame, the VMs placement is optimized to minimize energy consumption and to maximize performance [22]. Beloglazov et al. proposed some heuristics for dynamic adaption of VM allocation at runtime based on the current utilization of resources by applying live migration, switching idle nodes to sleep mode [3]. Goiri et al. presented an energy-efficient and multifaceted scheduling policy, modeling and managing a virtualized data center, in which the allocation of VMs is based on multiple facets to optimize the provider's profit [23]. Wang et al. investigated adaptive model-free approaches for resource allocation and energy management under time-varying workloads and heterogeneous multitier applications, and multiple metrics including throughput, rejection amount, queuing state were considered to design resource adjustment schemes [24]. Graubner et al. proposed an energy-efficient scheduling algorithm that was based on performing live migrations of virtual machines to save energy, and the energy costs of live migrations including pre- and post-processing phases were considered [25]. Unfortunately, to the best of our knowledge, seldom work considers the dynamic energy-efficient scheduling issue for real-time tasks in virtualized clouds. In this study, we focus on the energy-efficient scheduling by rolling-horizon optimization to efficiently guarantee the schedulability of real-time tasks and at the same time striving to save energy by dynamic VMs consolidation.

3 Problem Formulation

In this section, we will introduce the models, notions, and terminology used in this paper.

3.1 Scheduling Model

We target a virtualized cloud that is characterized by an infinite set $H = \{h_1, h_2, \ldots\}$ of physical computing hosts providing the hardware infrastructure for creating virtualized resources to satisfy users' requirements. The active host set is modeled by H_a with n elements, $H_a \subseteq H$. For a given host h_k, it is characterized by its CPU performance defined by Million Instructions Per Second (MIPS) [28], [30], amount of RAM, and network bandwidth, i.e., $h_k = \{c_k, r_k, n_k\}$, where c_k, r_k, and n_k represent the CPU capability, RAM and network bandwidth of the kth host, respectively. For each host h_k, it contains a set $V_k = \{v_{1k}, v_{2k}, \ldots v_{|V_k|k}\}$ of virtual machines (VMs). For a given VM v_{jk}, we use $c(v_{jk})$, $r(v_{jk})$, and $n(v_{jk})$ to denote the fractions of CPU performance, amount of RAM, and network bandwidth allocated to v_{jk}. Also, multiple VMs can be dynamically started and stopped on a single host

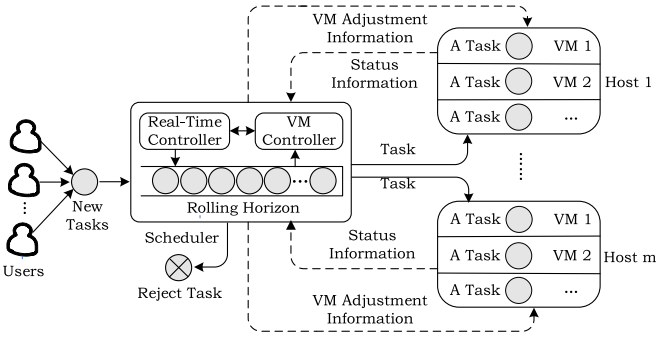


Fig. 1. Scheduling architecture.

based on the system workload. At the same time, some VMs are able to migrate across hosts in order to consolidate resources and further reduce energy consumption. Fig. 1 illustrates the scheduling architecture used for rolling-horizon optimization.

The scheduler consists of a rolling-horizon, a real-time controller, and a VM controller. The scheduler takes tasks from users and allocates them to different VMs. The rolling-horizon holds both new tasks and waiting tasks to be executed. A scheduling process is triggered by new tasks, and all the tasks in the rolling-horizon will be rescheduled.

When a new task arrives, the scheduling process follows five steps as below:

Step 1. The scheduler checks the system status information such as running tasks' remaining execution time, active hosts, VMs' deployments, and the information of tasks in waiting pool including their deadlines, currently allocated VMs, start time, etc.

Step 2. The tasks in rolling-horizon are sorted by their deadlines to facilitate the scheduling operation.

Step 3. The real-time controller determines whether a task in the rolling-horizon can be finished before its deadline. If not, the real-time controller informs the VM controller, and then the VM controller adds VMs to finish the task within timing constraint. If no schedule can be found to satisfy the task's timing requirement although enough VMs have been added, the task will be rejected. Otherwise, the task will be retained in the rolling-horizon.

Step 4. The scheduling decision for the tasks in the rolling-horizon is updated, e.g., their execution orders, start times, allocated VMs and new active hosts.

Step 5. When a task in the rolling-horizon is ready to execute, the task is dispatched to assigned VM.

Additionally, when tasks arrive slowly, tasks have loose deadlines or their count is less, making system workload light, the VM controller considers both the status of active hosts and the task information, and then decides whether some VMs should be stopped or migrated to consolidate resources so as to save energy.

Both of the real-time controller and the VM controller work together trying to first meet users' timing requirements and then reduce energy consumption by dynamically allocating tasks, adjusting VMs and hosts.

3.2 Task Model

In this study, we consider a set $T = \{t_1, t_2, \ldots\}$ of independent tasks that arrive dynamically. A task t_i submitted by

a user can be modeled by a collection of parameters, i.e., $t_i = \{a_i, l_i, d_i, f_i\}$, where a_i, l_i, d_i and f_i are the arrival time, task length/size, deadline, and finish time of task t_i, respectively. We let rt_{jk} be the ready time of VM v_{jk} at host h_k. Similarly, let st_{ijk} be the start time of task t_i on VM v_{jk}. Due to the heterogeneity in terms of CPU processing capabilities of VMs, we let et_{ijk} be the execution time of task t_i on VM v_{jk}:

$$et_{ijk} = \frac{l_i}{c(v_{jk})}. \tag{1}$$

It is assumed that ft_{ijk} is the finish time of task t_i on VM v_{jk}, and it can be easily determined as follows:

$$ft_{ijk} = st_{ijk} + et_{ijk}. \tag{2}$$

In addition, x_{ijk} is employed to reflect a mapping of tasks to VMs at different hosts in a virtualized cloud, where x_{ijk} is "1" if task t_i is allocated to VM v_{jk} at host h_k and is "0", otherwise.

The finish time is, in turn, used to determine whether the task's timing constraint can be guaranteed, i.e.,

$$x_{ijk} = \begin{cases} 0, & \text{if } ft_{ijk} > d_i, \\ 1 \text{ or } 0, & \text{if } ft_{ijk} \le d_i. \end{cases} \tag{3}$$

3.3 Energy Consumption Model

The energy consumption by hosts in a data center is mainly determined by CPU, memory, disk storage and network interfaces, in which the CPU consumes the main part of energy. So we consider in this paper the energy consumption by CPU like [3]. Further, the energy consumption can be classified into two parts, i.e., dynamic energy consumption and static (i.e., leakage) energy consumption [26]. Since the dynamic energy consumption is normally dominant and the static energy consumption follows a similar trend to the dynamic one, we focus on the dynamic energy consumption while building the energy consumption model.

Let ec_{ijk} be the energy consumption caused by task t_i running on VM v_{jk}. We denote the energy consumption rate of the VM v_{jk} by ecr_{jk} and the energy consumption ec_{ijk} can be calculated as follows:

$$ec_{ijk} = ecr_{jk} \cdot et_{ijk}. \tag{4}$$

Hence, the total energy consumed by executing all the tasks is:

$$\begin{aligned} ec^{exec} &= \sum_{k=1}^{|H_a|} \sum_{j=1}^{|V_k|} \sum_{i=1}^{|T|} x_{ijk} \cdot ec_{ijk} \\ &= \sum_{k=1}^{|H_a|} \sum_{j=1}^{|V_k|} \sum_{i=1}^{|T|} x_{ijk} \cdot ecr_{jk} \cdot et_{ijk}. \end{aligned} \tag{5}$$

We assume in Eq. (5) that no energy consumption is incurred when VMs are sitting idle. However, this assumption is not valid in real virtualized cloud data centers. The energy consumption when VMs are idle includes two parts, i.e., all the VMs in a host are idle and some of VMs in a host are idle.

When all the VMs in a host are sitting idle, this host can be set to a lower energy consumption rate by DVFS technology. Thus, in this case we denote the energy consumption rate of VM v_{jk} by ecr'_{jk}. Suppose the idle time when all the VMs in a host h_k are idle is it_k, the energy consumption when a host is idle (i.e., all the VMs in this host are idle) can be written as:

$$ec^{allIdle} = \sum_{k=1}^{|H_a|} \sum_{j=1}^{|V_k|} ecr'_{jk} \cdot it_k. \tag{6}$$

If only parts of VMs in a host are idle, the energy consumption rates of VMs are same as that when they are executing tasks, i.e., the energy consumption rate of VM v_{jk} is ecr_{jk}. Then, we obtain the analytical formula for the energy consumed in this case as:

$$\begin{aligned} ec^{partIdle} &= \sum_{k=1}^{|H_a|} \sum_{j=1}^{|V_k|} ecr_{jk} \cdot t_j^{partIdle} \\ &= \sum_{k=1}^{|H_a|} \sum_{j=1}^{|V_k|} ecr_{jk} \cdot \left(\max_{i=1}^{|T|} \{f_i\} - it_k - \sum_{i=1}^{|T|} x_{ijk} \cdot et_{ijk} \right), \end{aligned} \tag{7}$$

where $t_j^{partIdle}$ is the idle time of VM v_{jk} when only some VMs are sitting idle in host h_k.

Therefore, the energy consumption considering the execution time and idle time is derived from Eq. (5), Eq. (6), and Eq. (7) as:

$$\begin{aligned} ecei &= ec^{exec} + ec^{allIdle} + ec^{partIdle} \\ &= \sum_{k=1}^{|H_a|} \sum_{j=1}^{|V_k|} \sum_{i=1}^{|T|} x_{ijk} \cdot ecr_{jk} \cdot et_{ijk} \\ &\quad + \sum_{k=1}^{|H_a|} \sum_{j=1}^{|V_k|} ecr'_{jk} \cdot it_k \\ &\quad + \sum_{k=1}^{|H_a|} \sum_{j=1}^{|V_k|} ecr_{jk} \cdot \left(\max_{i=1}^{|T|} \{f_i\} - it_k - \sum_{i=1}^{|T|} x_{ijk} \cdot et_{ijk} \right). \end{aligned} \tag{8}$$

Noticeably, the host may not be fully utilized, e.g., although some VMs are placed on a host, some of the host resource is still unused. However, the resource also consume energy. Suppose there are s periods in each which the count of VMs in host h_k is different from another. We use t_p to denote the time in the period p, then we can get the following energy consumption by unused resource.

$$ecur = \sum_{k=1}^{|H_a|} \sum_{p=1}^{s} \left(ecr(h_k) - \sum_{j=1}^{|V_k(p)|} ecr_{jk} \right) \cdot t_p, \tag{9}$$

where $|V_k(p)|$ denotes the VMs' count in the pth period of host h_k.

Consequently, the total energy consumption to execute all the allocated tasks can be derived from Eq. (8) and Eq. (9) as:

$$ec = ecei + ecur. \tag{10}$$

From the aforementioned analysis of energy consumption, we can get that the less running hosts, the less consumed energy. However, less hosts may greatly affect the guarantee ratio (GR) of real-time tasks. It can be known that energy conservation and tasks' guarantee ratio are two conflicting objectives while scheduling in a virtualized cloud. Our EARH scheduling strategy makes a good trade-off between guarantee ratio and energy saving by dynamically starting hosts, closing hosts, creating VMs, canceling VMs and migrating VMs according to the system workload.

4 The EARH Scheduling Strategy

In this section, we first introduce the methodology of rolling-horizon optimization and then integrate an energy-efficient scheduling algorithm into it.

4.1 Rolling-Horizon Optimization

Unlike the traditional scheduling scheme where once a task is scheduled, it is dispatched immediately to the local queue of a VM or a host, our approach puts all the waiting tasks in a rolling horizon and their schedules are allowed to be adjusted for the system schedulability and possibly less energy consumption. The pseudocode of RH optimization is shown in Algorithm 1.

Algorithm 1 Pseudocode of RH Optimization

```
1: for each new task t_i do
2:    Q ← NULL; R ← NULL;
3:    Add a new task t_i into set Q;
4:    for each task t_w do
5:       if st_wjk > a_i and x_wjk == 1 then
6:          Add task t_w into set Q;
7:       end if
8:       if st_wjk + et_ijk ≥ rt_jk and x_wjk == 1 then
9:          rt_jk ← st_wjk + et_wjk;
10:         Update the ready time of v_jk in Vector R;
11:      end if
12:      Sort tasks in Q by their deadlines in a non-descending
         order;
13:      for each task t_q in set Q do
14:         Schedule task t_q by Different Energy-Efficient
            Scheduling Algorithms;
15:         if x_qjk == 0 then
16:            Reject task t_q;
17:         end if
18:      end for
19:      Update scheduling decisions;
20:   end for
21: end for
```

In the pseudocode of RH optimization, a new task is added into set Q which represents a rolling-horizon when the new task arrives (see line 3). Then, it adds all the waiting tasks in the set Q (see lines 5-7). The ready time of each VM is updated (see lines 8-11). And it sorts the tasks in Q by their deadlines (see line 12). After that, the tasks in Q can be scheduled using different algorithms and if a task cannot be allocated, then the task will be rejected (see lines 13-18).

4.2 Energy-Aware Scheduling Algorithm

In our energy-aware scheduling algorithm, we attempt to append a new task to the end of former allocated tasks on a VM. So the start time st_{ijk} of task t_i on VM v_{jk} can be calculated as below:

$$st_{ijk} = \max\{rt_{jk}, a_i\}, \tag{11}$$

where rt_{jk} is the ready time of VM v_{jk}, and it is updated when each task is allocated to v_{jk}, e.g., a new task t_p is allocated to v_{jk}, the new ready time rt_{jk} of v_{jk} is:

$$rt_{jk} = st_{pjk} + et_{pjk}. \tag{12}$$

The pseudocode of energy-efficient scheduling algorithm is shown in Algorithm 2.

Algorithm 2 Pseudocode of energy-efficient scheduling

```
1: findTag ← FALSE; findVM ← NULL;
2: for each VM v_jk in the system do
3:    Calculate the start time st_ijk by Eq. (11) and the execu-
      tion time et_ijk by Eq. (1);
4:    if st_ijk + et_ijk ≤ d_i then
5:       findTag ← TRUE;
6:       Calculate ec_ijk by Eq. (4);
7:    end if
8: end for
9: if findTag == FALSE then
10:   ScaleUpResource();
11: end if
12: if findTag == TRUE then
13:   Select v_sk with minimal energy consumption to execute
      t_i; findVM ← v_sk; x_ijk ←1;
14: else
15:   x_ijk ←0;
16: end if
17: Update the scheduling decision of t_i and remove it from
    Q;
```

The energy-efficient scheduling algorithm is in heuristic fashion. It allocates each task to a VM in a way to aggressively meet tasks' deadlines while conserving energy consumption. The energy-efficient scheduling algorithm calculates the task t_i's start time and execution time on each VM (see line 3). If t_i's deadline can be satisfied representing this task can be allocated, then the algorithm calculates t_i's energy consumption (see lines 4-7). If t_q cannot be successfully allocated to any current VMs, it calls the Function ScaleUpResource() striving to accommodate t_i by increasing resource (see lines 9-11). If t_i can be allocated, then it selects the VM yielding minimal energy consumption to execute the task, otherwise, the algorithm rejects t_i (see lines 12-16).

When a task cannot be successfully allocated in any current VM, the ScaleUpResource() is called to create a new VM with the goal of finishing the task within its deadline. In our study, we employ a three-step policy to create a new VM as follows:

Step 1. Create a new VM in a current active host without any VM migration;

Step 2. If a new VM cannot be created in Step 1, migrate some VMs among current active hosts to yield enough resource on a host and then create a VM on it;

Step 3. If a new VM cannot be created in Step 2, start a host and then create a new VM on it.

We use $st(h_k)$, $ct(v_{jk})$, and $mt(v_{jk})$ to denote the start-up time of host h_k, the creation time of VM v_{jk} and the migration time of VM v_{jk}, respectively. The migration time $mt(v_{jk})$ can be defined as [27]:

$$mt(v_{jk}) = \frac{r(v_{jk})}{n(v_{jk})}. \tag{13}$$

It should be noted that using different steps products different start times for a task, i.e.,

$$st_{ijk} = \begin{cases} a_i + ct(v_{jk}), & \text{if step 1,} \\ a_i + ct(v_{jk}) + \sum_{p=1}^{|p|} mt(v_{pk}), & \text{if step 2,} \\ a_i + st(h_k) + ct(v_{jk}), & \text{if step 3.} \end{cases} \tag{14}$$

The pseudocode of Function ScaleUpResource() is shown in Algorithm 3.

Algorithm 3 Pseudocode of Function ScaleUpResource()

1: Select a kind of VM v_j with minimal MIPS on condition that t_i can be finished before its deadline;
2: Sort the hosts in H_a in the decreasing order of the CPU utilization;
3: **for** each host h_k in H_a **do**
4: **if** VM v_j can be added in host h_k **then**
5: Create VM v_{jk}; $findTag \leftarrow$ TRUE; break;
6: **end if**
7: **end for**
8: **if** $findTag ==$ FALSE **then**
9: Search the host h_s with minimal CPU utilization;
10: Find the VM v_{ps} with minimal MIPS in h_s;
11: **for** each host h_k except h_s in H_a **do**
12: **if** VM v_{ps} can be added in host h_k **then**
13: Migrate VM v_{ps} to host h_k; break;
14: **end if**
15: **end for**
16: **if** VM v_j can be added in host h_s **then**
17: Create VM v_{js};
18: **if** t_i can be finished in v_{js} before its deadline **then**
19: $findTag \leftarrow$ TRUE;
20: **end if**
21: **end if**
22: **end if**
23: **if** $findTag ==$ FALSE **then**
24: Start a host h_n and put it in H_a;
25: Create VM v_{jn} on h_n;
26: **if** t_i can be finished in v_{jn} before its deadline **then**
27: $findTag \leftarrow$ TRUE;
28: **end if**
29: **end if**

In Function ScaleUpResource(), our algorithm first selects a kind of VM v_j (it has not been placed on a host) that can finish the task within its deadline (see line 1). Then it selects a host with possibly minimal remaining MIPS that can accommodate v_j (see lines 2-7). If no such host can be found, it migrates the VM with minimal MIPS on the host with minimal CPU utilization to a host with possibly minimal remaining MIPS (see lines 9-15). After this migration, it checks whether the VM v_j can be added to the host on which a VM has been migrated. If so, the v_j will be created on the host and it checks if the task can be finished on v_j before the task's deadline (see lines 16-21). If no migration is feasible or the task cannot be successfully finished, then it starts a host h_n and creates v_{jn} on it. Then it checks if the task can be finished successfully on v_{jn} (see lines 23-29).

Theorem 1. *The time complexity of our energy-aware task allocation algorithm is $O(N_tN_{vm} + N_tN_alog(N_a))$, where N_a is the number of active hosts, N_{vm} is the number of VMs, N_t is the number of tasks.*

Proof. The time complexity of calculating a task's start time and execution time on all the VMs in the system is $O(N_{vm})$ (lines 3-9, Algorithm 2). In Algorithm 3, the complexity of selecting a VM with minimal MIPS is $O(N_{vm})$ (line 1, Algorithm 3). It takes $O(N_alog(N_a))$ to sort hosts in a decreasing order (line 2, Algorithm 3). Checking if a VM can be added to a host, the time complexity is $O(N_a)$ (lines 3-7, Algorithm 3). The time complexity of finding the host with minimal CPU utilization is $O(N_a)$ (line 9, Algorithm 3). It takes $O(N_{vm})$ to find the VM with minimal MIPS (line 10, Algorithm 3). Checking if a VM can be migrated to a host, the time complexity is $O(N_a)$ (lines 11-15, Algorithm 3). For other lines, the time complexity is $O(1)$. Hence, the time complexity of our energy-aware task allocation algorithm is calculated as $O(N_t)(O(N_{vm}) + O(N_alog(N_a)) + O(N_a)) = O(N_tN_{vm} + N_tN_alog(N_a))$. □

When the VMs do not use all the provided resources, they can be logically resized and consolidated to the minimal count of physical hosts, while idle hosts can be shut down to eliminate the idle energy consumption and thus reduce the total energy consumption by the cloud. Algorithm 4 gives the pseudocode of algorithm about scaling down resources.

Algorithm 4 Pseudocode of Scaling Down Algorithm

1: $SH \leftarrow \emptyset$; $DH \leftarrow \emptyset$;
2: **for** each VM v_{jk} in the system **do**
3: **if** v_{jk}'s idle time $it_{jk} >$ THRESH **then**
4: Remove VM v_{jk} from host h_k and delete it;
5: **end if**
6: **end for**
7: **for** each host h_k in H_a **do**
8: **if** there is no VM on h_k **then**
9: Shut down host h_k and remove it from H_a;
10: **end if**
11: **end for**
12: Sort the hosts in H_a in an increasing order of the CPU utilization;
13: $SH \leftarrow H_a$; $DH \leftarrow H_a$ and sort DH inversely;
14: **for** each host h_k in SH **do**
15: $shutDownTag \leftarrow$ TRUE; $AH \leftarrow \emptyset$;
16: **for** each VM v_{jk} in h_k **do**
17: $migTag \leftarrow$ FALSE;
18: **for** each host h_p in DH except h_k **do**
19: **if** v_{jk} can be added in h_p **then**
20: $migTag \leftarrow$ TRUE; $AH \leftarrow h_p$; break;
21: **end if**
22: **end for**
23: **if** $migTag ==$ FALSE **then**
24: $shutDownTag \leftarrow$ FALSE; break;
25: **end if**
26: **end for**
27: **if** $shutDownTag \leftarrow$ TRUE **then**
28: Migrate VMs in h_k to destination hosts; $SH \leftarrow SH - AH - h_k$; $DH \leftarrow DH - h_k$;
29: Shut down host h_k and remove it from H_a;
30: **end if**
31: **end for**

In the above algorithm, if there exists any VM whose idle time is larger than a preestablished threshold, then this VM will be canceled (see lines 2-6). After this operation, if a host has no VMs running on it, then it shuts down this host (see lines 7-11). Algorithm 4 puts hosts into sets SH and DH, respectively. In the SH, the hosts are sorted by their CPU utilizations in an increasing order, and DH is opposite (see lines 12 and 13). If all the VMs running on a host that is in SH can be added to one or some hosts in DH, it migrates these VMs to destination hosts, and then shuts down the host after this migration. Otherwise, if in the host there are one or some VMs that cannot be migrated, then it gives up the migration operation of all the VMs on the host. At the same time, the destination hosts will be removed from set SH and the host that is shut down will also be removed from DH (see lines 14-31).

Theorem 2. *The time complexity of our scaling down algorithm is $O(N_a^2 N_{vm})$, where N_a is the number of active hosts, N_{vm} is the number of VMs.*

Proof. Checking whether there exists one VM whose idle time is larger than a preestablished threshold, the time complexity is $O(N_{vm})$ (lines 2-6, Algorithm 4). It takes $O(N_a)$ to check if a host needs to be shut down (lines 7-11, Algorithm 4). It takes $O(N_a log(N_a))$ to sort hosts in an increasing order (line 12, Algorithm 4). The VM migration takes $O(N_a^2 N_{vm})$ (lines 14-31). For other lines, the time complexity is $O(1)$. Therefore, the complexity of our scaling down algorithm is calculated as $O(N_{vm}) + O(N_a) + O(N_a log(N_a)) + O(N_a^2 N_{vm}) = O(N_a^2 N_{vm})$. □

5 Performance Evaluation

To demonstrate the performance improvements gained by EARH, we quantitatively compare it with three baseline algorithms—non-RH-EARH (NRHEARH in short), non-Migration-EARH (NMEARH in short), and non-RH-Migration-EARH (NRHMRARH in short). In addition, we also compare them with four existing algorithms—ProfRS in [28], Greedy-R in [29], Greedy-P in [29] and FCFS in [29]. The algorithms for comparison are briefly described as follows:

NRHEARH. Differing from EARH, NRHEARH does not employ the rolling-horizon optimization.

NMEARH. Differing from EARH, NMEARH does not employ the VM migration while allocating real-time tasks.

NRHMEARH. Differing from EARH, NRHMEARH employ neither the rolling-horizon optimization nor the VM migration.

ProfRS. It first checks if a new task can wait until all the accepted tasks complete in any initiated VMs. If the new task cannot wait, then it checks whether the new task can be inserted before any accepted tasks in any initiated VMs. If not, the algorithm checks if the new task can be accepted by initiating a new VM. This algorithm does not consider consolidating VMs to minimal number of servers when the system workload is light.

Greedy-R. It assigns tasks with the quickest execution time first to the most powerful available cloud resource in order to maximize system response time.

TABLE 1
Parameters for Simulation Studies

Parameter	Value (Fixed)-(Varied)
Task Count (10^5)	(1)-(0.5,1.0,1.5,2.0,2.5,3.0)
$baseTime$ (s)	(250)-(100,150,200,250,300,350,400)
$intervalTime$ (s)	(3)-(0,2,4,6,8,10,12)
$taskLength$ (MI)	(100000)

Greedy-P. It assigns tasks with the quickest execution time first to the less powerful available cloud resource so as to maximize task parallelization as well as system response time.

FCFS. It assigns tasks as soon as they are ready for execution to any available cloud resource.

The performance metrics by which we evaluate the system performance include:

1. Guarantee ratio defined as: GR = Total count of tasks guaranteed to meet their deadlines/Total count of tasks.
2. Total energy consumption is: Total energy consumed by hosts.
3. Energy consumption per task (ECT) calculated as: ECT= Total energy consumption/Accepted task count.
4. Resource utilization: The average host utilization, which can be calculated as: $RU = (\sum_{k=1}^{|H_a|} \sum_{j=1}^{|V_k|} \sum_{i=1}^{|T|} l_i \cdot x_{ijk})/(\sum_{k=1}^{|H_a|} c_k \cdot wt_k)$, where wt_k is the active time for host h_k during an experiment.

5.1 Simulation Method and Parameters

In order to ensure the repeatability of the experiments, we choose the way of simulation to evaluate the performance of aforementioned algorithms. In our simulations, the CloudSim toolkit [30] is chosen as a simulation platform, and we add some new settings to conduct our experiments. The detailed setting and parameters are given as follows:

1. Each host is modeled to have one CPU core and the CPU performance is 1,000 MIPS, 1,500 MIPS, or 2,000 MIPS.
2. The energy consumption rate of the three different kinds of hosts are 200, 250, or 400 W.
3. The start-up time of a host is 90 s and the creation time of a VM is 15 s.
4. We employ parameter $baseDeadline$ to control a task's deadline that can be calculated as:

$$d_i = a_i + baseDeadline, \tag{15}$$

   where parameter $baseDeadline$ is in uniform distribution $U(baseTime, a \times baseTime)$ and we set $a = 4$.
5. The arriving rate of tasks is in Poisson distribution, and the parameter $intervalTime$ is used to determine the time interval between two consecutive tasks.

The values of parameters are listed in Table 1.

5.2 Performance Impact of Task Count

In this section, we present a group of experimental results to observe the performance comparison of the eight algorithms in terms of the impact of task count. Fig. 2 shows the experimental results.

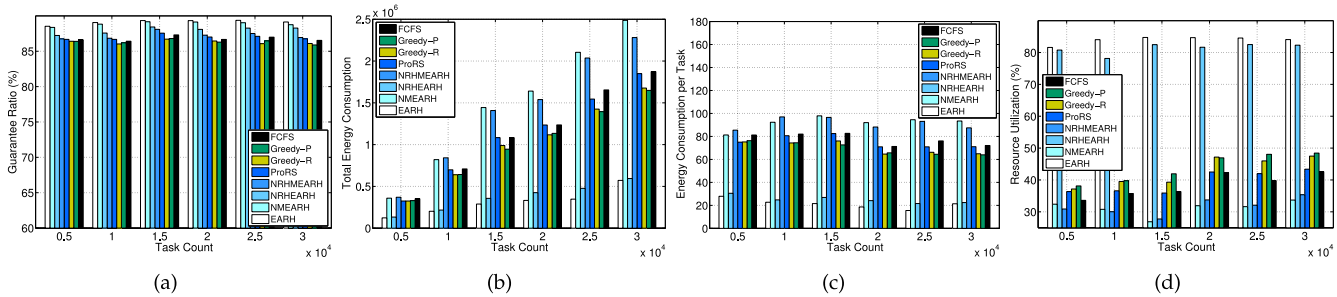


Fig. 2. Performance impact of task count.

We can observe from Fig. 2a that all the algorithms basically keep stable guarantee ratios regardless of how the changes of task count. This is because there are infinite resources in clouds, thus when task count increases, new hosts will be started up to finish more tasks. However, not all the tasks can be finished successfully although there are enough resources. We can attribute this to the fact that starting a new host or creating a new VM needs additional time cost, which makes some real-time tasks with tight deadlines cannot be finished within their timing constraints. Besides, it can be found that EARH and NMEARH have higher guarantee ratios than the other six algorithms. This can be explained that EARH and NMEARH employ the rolling-horizon optimization policy that is able to make tasks with tight deadlines finish earlier, so the schedulability is significantly improved.

From Fig. 2b, it can be observed that although NRHEARH conserves more energy, its guarantee ratio is bad (see Fig. 2a); in contrast, NMEARH has the most energy consumption but with higher guarantee ratio, which reflects that NRHEARH and NMEARH lack good trade-off between guarantee ratio and total energy consumption. Moreover, we can see that EARH and NRHEARH have better energy conservation ability compared with others, and the trend becomes more obvious with the increase of task count. This experimental result indicates that employing the VM migration policy is very efficient when scheduling real-time tasks. On one hand, when the task count increases, current VMs can be consolidated to make some room for creating new VMs, which avoids the energy consumption caused by adding new active hosts. On the other hand, the VMs in light-load host can be migrated to other hosts and then the idle hosts can be shut down, which further reduces the energy consumption.

Fig. 2c reveals that the ECTs of NMEARH and NRHMEARH slightly increase with the increase of task count, whereas other six algorithms that employ VM migration policy basically maintain stable ECT. This experimental result can be explained in two folds. First, when the task count increases, new VMs are needed to accommodate these real-time tasks. The RAEH and NRHEARH use the migration policy striving to make room for theses VMs, and avoid starting new hosts. Therefore, the current resource is efficiently utilized leading to a basically stable value of ECT. Second, ProRS, Greedy-R, Greedy-P and FCFS tend to reject the tasks with tight deadlines avoiding to increase the usage of resources.

Fig. 2d demonstrates that EARH and NRHEARH show significant advantage compared with the other algorithms. This can be attributed to the benefit brought by VM migration policy. It is the VM migration policy that can make the system fully utilize the host computing capacity. On average, EARH outperforms NMEARH and NRHMEARH 170.4 and 167.1 percent, respectively. Compared with ProRS, Greedy-R, Greedy-P and FCFS, EARH outperforms them by 114.1, 98.0, 92.9 and 120.2 percent on average, respectively.

5.3 Performance Impact of Task Arrival Rate

In order to examine the performance impact of task arrival rate, we vary the value of $intervalTime$ from 0 to 12 with Step 2. Fig. 3 illustrates the performance of the eight algorithms.

One of the observations from Fig. 3a is that the eight algorithms basically have unchanged guarantee ratios no matter how the $intervalTime$ varies. This derives from the infinite resource provided in clouds. When the value of $intervalTime$ is smaller, tasks arrive in short time. In this situation, creating new VMs or starting hosts is required to accommodate these tasks. In addition, EARH and NMEARH employing rolling-horizon optimization have higher guarantee ratios than other six algorithms without

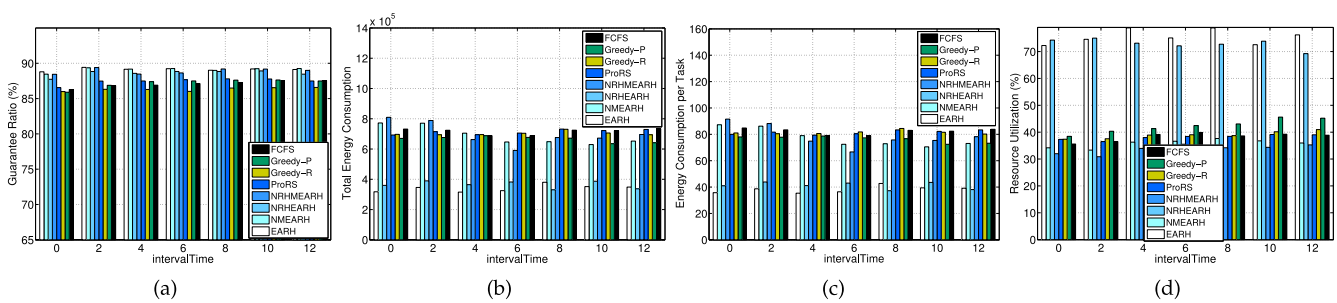


Fig. 3. Performance impact of task arrival rate.

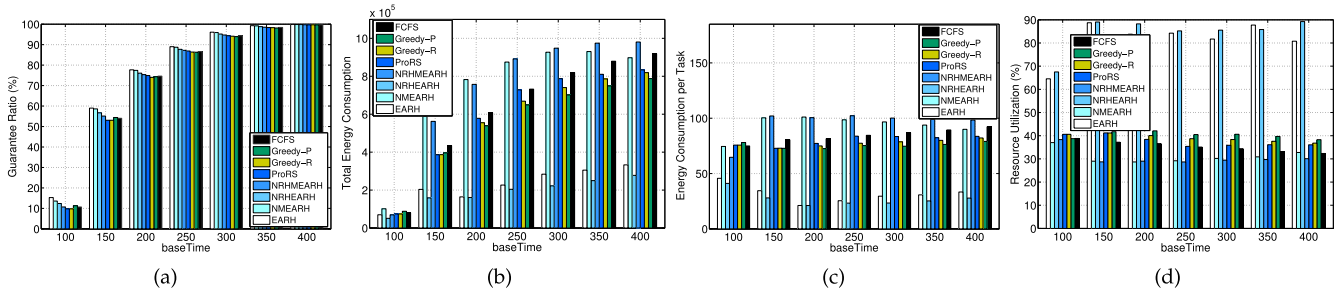


Fig. 4. Performance impact of task deadline.

using rolling-horizon. The explanation for this experimental result is like that in Fig. 2a.

Fig. 3b shows that EARH and NRHEARH consume less energy than the other algorithms. This reason can be explained as that in Fig. 2b. Besides, we can get from Fig. 3b that when the parameter $intervalTime$ changes, the energy consumptions by the eight algorithms are basically unchangeable indicating that the task arrival rate has little impact on energy consumption.

The experimental results from Fig. 3c depict that when $intervalTime$ changes, the ECTs of the eight algorithms are basically constant demonstrating that task arrival rate has little impact on ECT. Besides, the ECTs of EARH and NMEARH are obviously less than those of the other algorithms with the explanation like that in Fig. 2c. In addition, when tasks arrive almost at the same time (e.g., $intervalTime$ is 0 or 2), NMEARH and NRHMEARH consume more energy per task than ProRS, Greedy-R, Greedy-P and FCFS. This is because NMEATH and NRHMEARH accept more tasks with tight deadlines than the others. When these tasks arrive continuously, the system has to create new VMs and hosts to accommodate them, which increases the energy consumption by the tasks with tight deadlines.

Fig. 3d shows that when the $intervalTime$ varies, EARH and NRHEARH also exhibit their advantage brought by VM migration policy. On average, EARH outperforms NMEARH and NRHMEARH 111.1 and 120.2 percent, respectively. Compared with ProRS, Greedy-R, Greedy-P and FCFS, EARH outperforms them by 92.7, 87.4, 69.4 and 94.7 percent on average, respectively.

5.4 Performance Impact of Task Deadline

The goal of this set of experiments is to evaluate the performance impact of task deadline on the eight algorithms. Parameter $baseTime$ varies from 100 to 400 with step 50.

It is observed from Fig. 4a that with the increase of $baseTime$ (i.e., task deadline becomes looser), the guarantee ratios of the eight algorithms increase correspondingly. This is because the deadlines are prolonged making tasks can be finished later within their timing constraints. In addition, Fig. 4a shows that EARH and NMEARH have higher guarantee ratios. This can be explained that after employing the rolling-horizon policy, the tasks with tight deadlines can be preferentially finished. Thus, the guarantee ratio is enhanced.

From Fig. 4b, we can see that when $baseTime$ increases, the energy consumption by the eight algorithms increases correspondingly. This is because when the deadline becomes looser, more tasks can be finished before their deadlines and thus more energy is consumed. Also, the energy consumptions by EARH and NRHEARH are less than those of others and the trend becomes more pronounced with the increase of $baseTime$. We attribute this trend to the fact that EARH and NRHEARH use the VM migration policy that can efficiently utilize the resource of active hosts, which avoids starting more hosts to finish tasks. However, the other six algorithms need to constantly start hosts to finish more tasks resulting in more energy consumption.

Fig. 4c shows that the ECTs of NMEARH, NRHMEARH, ProRS, Greedy-R, Greedy-P and FCFS become larger with the increase of $baseTime$. This can be explained that these algorithms start more hosts and thus yield more idle resource leading to lower utilization. When the value of $baseTime$ is less than 300, the ECTs of EARH and NRHEARH decrease with the explanation that when deadline becomes looser, more tasks can be finished in the current active hosts. Besides, the VM migration policy is employed without starting more hosts. Hence, the utilization of active hosts is higher leading to smaller ECT. Nevertheless, when the value of $baseTime$ is larger than 300, the current active hosts lack the ability to finish more tasks due to looser deadline and some hosts must be started, so yielding some idle resource. Therefore, the ECTs increase correspondingly.

The advantage of VM migration policy is again shown in Fig. 4d. EARH and NRHEARH have much higher resource utilizations than the other algorithms. Besides, we can see that the resource utilization of NRHEARH is sometimes even higher than EARH. It can be explained that the employment of rolling-horizon policy makes the system accept more tasks with tight deadlines which sometimes needs new computing resources, and thus decreases the resource utilization a bit. EARH outperforms NMEARH and NRHMEARH 164.1 and 173.1 percent, respectively. Compared with ProRS, Greedy-R, Greedy-P and FCFS, EARH outperforms them by 124.1, 118.1, 109.6 and 143.2 percent on average, respectively.

5.5 Evaluation Based on Real-World Trace

The above groups of experiments demonstrate the performance of the different algorithms in various random synthetic workloads. To validate our proposed algorithm in practical use, we evaluate the algorithms based on the latest version of the Google cloud tracelogs [31].

The tracelogs record the information of 25 million tasks grouped in 650 thousand jobs that span 29 days. It is

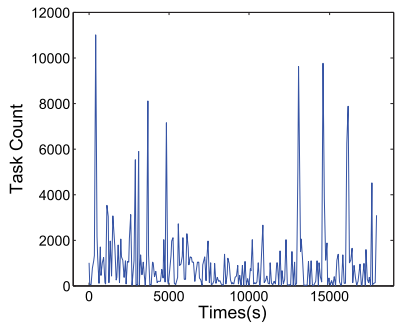


Fig. 5. The count of tasks submitted to the system.

relatively difficult to conduct an experiment based on all the tasks due to the enormous count of tasks in the tracelogs. As a result, the first 5 hours in day 18, a representative day among the 29 days according to the analysis in [34], were selected as a testing sample. Over 200 thousand tasks were submitted to the cloud system over this 5 hours. To observe the change of the task count over the time, we depict the count of tasks submitted in every 60 seconds in Fig. 5, where it can be easily found that the task count fluctuates significantly over the time. When large amounts of tasks surge into the system, the resource demand is at a peak. While the resource demand decreases sharply at the timestamps where a few of tasks were submitted into the system. Based on this observation, it is straightforward to conclude that the resource scaling-up and scaling-down are quite necessary for a cloud system.

It takes about 1,587 seconds averagely from the submission of a task to the finish of the task (i.e., the response time of a task). The average execution time of the tasks is around 1,198 seconds. Further, both the distributions of the response time and the execution time approximately follow Lognormal distribution, which means that most tasks finish in a short time. On average, the ratio of the tasks' response time over the execution time is 2.89.

Due to the lack of some context information and normalized data, it is necessary to make four realistic assumptions as follows:

- When a task fails (i.e., it is evicted or killed), we assume that it is reset back to the initial state. According to the statement in [32], [33], the Google cloud system manages to reschedule failed tasks, and restart the tasks from the initial state.
- The task execution duration is considered from the last schedule event to the finish event. Tasks are normally resubmitted and rescheduled caused by eviction and failures.
- Task length l_i is calculated based on the execution duration and the average CPU utilization. As the tracelog does not contain the data of task length in MI, we employ the method proposed in [34] to solve the problem:

$$l_i = (ts_{finish} - ts_{schedule}) \times U_{avg} \times C_{CPU}, \tag{16}$$

where ts_{finish} and $ts_{schedule}$ represent the timestamp of finish and schedule event, respectively; U_{avg} denotes the average CPU usage of this task. All the three values can be obtained in the tracelog. As regards C_{CPU}, it represents the processing capacity of the CPU in Google cloud. Since the data of machines' capacity is rescaled in the trace, we assume that it is similar to our experiment settings for hosts, $C_{CPU} = 1500$MIPS.
- The deadline of each task is designated through the ratio of response time over execution time. As discussed above, for the tasks in the first 5 hours in the day 18, the average ratio is 2.89. So the deadline of each task is assumed to be β times larger than its maximum execution time, where β is uniformly distributed in the range of [2.6, 3.2].

Table 2 shows the performance of the eight algorithms based on the Google cloud tracelogs. The results show that EARH exhibits an outstanding performance in real practice. The guarantee ratios of the eight algorithms all show high values. However, there are still some tasks that are rejected. This is because of the restriction of deadline in our experiment, while in Google there is no hard deadline restriction. The *ECT*s of the eight algorithms are less than those in the previous synthetic workloads. It is reasonable because there are many short-run tasks in the tracelogs; the task lengthes of these tasks are relatively small which consumes less energy. Regarding the resource utilization, the algorithms using VM migration policy still outperform the other algorithms. This result indicates that our proposed algorithm can enhance the system's resource utilization effectively in practice. Based on the Google Cloud workloads, in terms of resource utilization, EARH outperforms NMEARH, NRHEARH and NRHMEARH 67.3, 11.0 and 83.0 percent, respectively. Considering the other four algorithms, the average outperformance of EARH reaches 30.3 percent.

In order to further investigate how our proposed algorithm can improve the system utilization, we depict the change of active hosts' count (*AHC* in short) over time in Fig. 6, and compare EARH with the other algorithms respectively. The blue solid line represents the task count whose value is specified by the right side *Y*-axis. The red solid line and dashed lines represent the *AHC*s of EARH and the other algorithms whose value is specified by the left side *Y*-axis. It can be found from the *AHC* of EARH that the resource provisioning in the system is elastic according to the request demand. When large amounts of tasks surge

TABLE 2
Performance for Google Cloud Workloads

Metric \ Algorithm	EARH	NMEARH	NRHEARH	NRHMEARH	ProRS	Greedy-R	Greedy-P	FCFS
GR	96.7%	95.4%	94.1%	93.4%	95.0%	93.2%	97.0%	90.1%
TEC($\times 10^6$)	2.55	3.28	2.68	3.87	2.65	2.59	3.13	2.82
ECT	10.95	14.28	11.83	17.21	11.59	11.54	13.40	13.00
RU	72.8%	43.5%	65.6%	39.8%	61.9%	57.0%	50.7%	55.1%

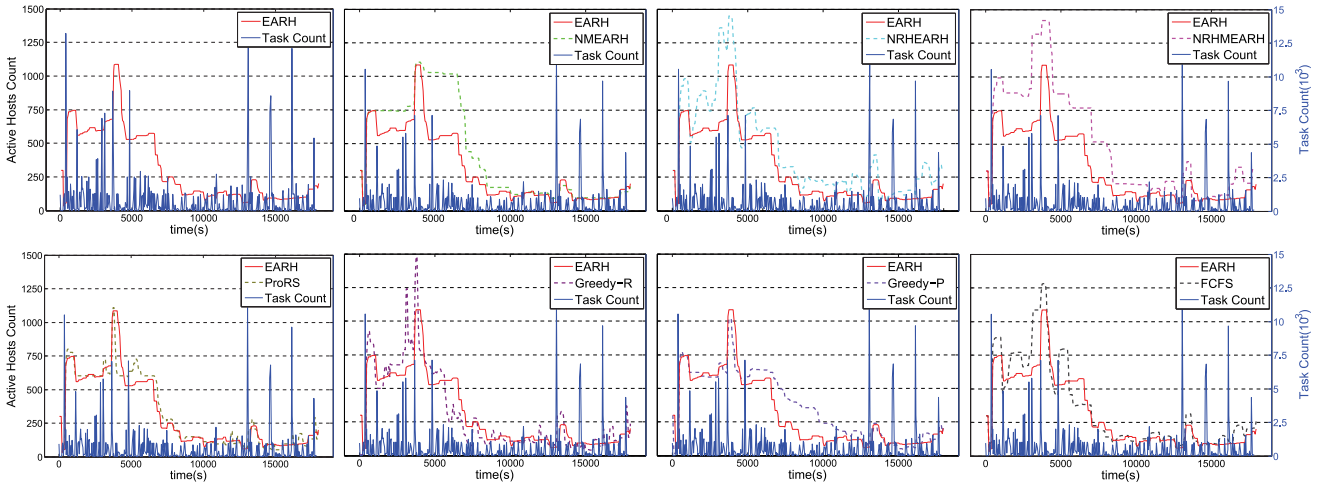


Fig. 6. The change of *AHC* and Task Count over time.

into the system, the *AHC* increases to accommodate these tasks. When fewer tasks are submitted to the system, the resource is over-provisioned, and the *AHC* decreases to save energy. From the comparison of EARH and NMEARH, we can observe that the system using EARH starts up less active hosts than NMEARH especially when the workload of the system changes from heavy to light. For example, during the time span from 5,000 to 10,000, the tasks submitted to system are much fewer than that in the previous time, the *AHC* of EARH decreases sharply while that of NMEARH still keeps a relatively high value. The superiority of EARH in this situation can be attributed to the VM migration technique which consolidates the existing VMs to several active hosts, and as a result the idle active hosts can be turned off to save energy when the tasks submitted to the system becomes fewer. The comparison of EARH and NRHEARH indicates that the employment of the rolling-horizon policy can reduce the demand of active hosts when large amounts of tasks surge into the system. It is because the adoption of the rolling-horizon policy is able to make the tasks with tight deadlines execute earlier, and postpone the execution of the tasks with loose deadlines. As a result, the resource demand in the time stamp at which large amounts of tasks submitted to the system become less, and the *AHC* of EARH is fewer than that of NRHEARH. For other four algorithms, the *AHC* of ProRS is similar to that of EARH; the Greedy-R starts up more active hosts when large amounts of tasks surge into the system; the Greedy-P cannot turn off the active hosts in time when fewer tasks are submitted to the system; the *AHC* of FCFS fluctuates with the change of task count.

As mentioned above, the information about tasks' deadlines is not contained in the Google tracelogs. We designate the deadline through the ratio of response time over execution time based on the realistic analysis in previous experiments. For the purpose of demonstrating the performance impact of deadline under the Google tracelogs, we vary the value of β from 1 to 5. Fig. 7 shows the experimental results.

It is observed from Fig. 7a that with the increase of β, the guarantee ratios of the eight algorithms increase. Besides, EARH and NMEARH have the higher guarantee ratios. This can be attributed to the employment of the rolling-horizon policy that enhances the guarantee ratio. Besides, it can be found that the rolling-horizon policy is especially suitable for tasks with tight deadlines, such as $\beta = 1$. From Fig. 7b, we can see that when β varies, the energy consumption by the eight algorithms decreases. This is because when the deadline is tight, the system has to start up more hosts to keep a high guarantee ratio, and thus consumes more energy. Also, the energy consumptions by EARH and NRHEARH are less than that of others by the adoption of the VM migration policy that can efficiently utilize the resource of active hosts. Fig. 7c shows that the *ECT*s of NMEARH, NRHMEARH, ProRS, Greedy-R, Greedy-P and FCFS decrease with the increase of β. This because these algorithms start up more hosts and thus incur a poor resource utilization. In addition, the descending trend of

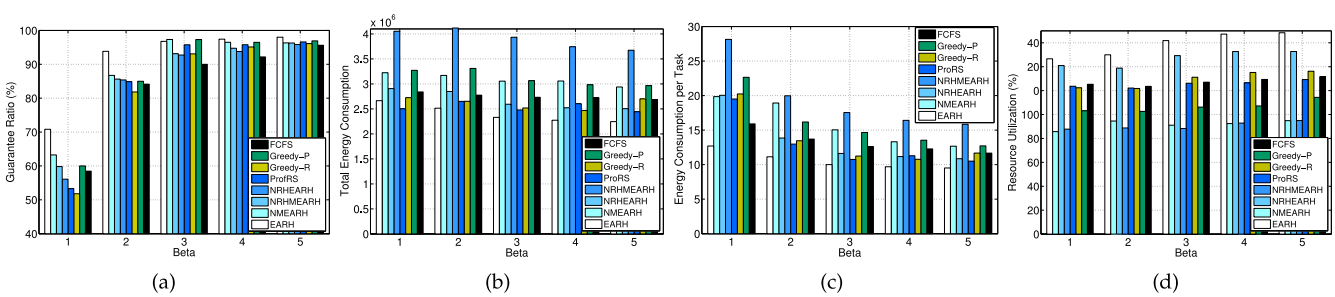


Fig. 7. Performance impact of Beta.

EARH is not as obvious as those of other algorithms. It is reasonable that EARH employs the rolling-horizon policy and the VM migration policy that make the system yield a good resource utilization even when the β is small. The advantage of the VM migration policy is again demonstrated in Fig. 7d. EARH and NRHEARH have higher resource utilizations than the other algorithms under the five values of β. EARH outperforms NMEARH, NRHEARH, and NMRHEARN 75.3, 8.1, and 76.3 percent respectively. For ProRS, Greedy-R, Greedy-P and FCFS, the average outperformance of EARH is 23.4, 26.0, 40.2, and 24.6 percent, respectively.

6 Conclusions and Future Work

In this paper, we investigated the problem of energy-aware scheduling for independent, aperiodic real-time tasks in virtualized clouds. The scheduling objectives are to improve the system's schedulability for real-time tasks and save energy. To achieve the objectives, we employed the virtualization technique and a rolling-horizon optimization scheme. First, we proposed a rolling-horizon scheduling architecture, and then a task-oriented energy consumption model was analyzed and built. On this basis, we presented a novel energy-aware scheduling algorithm named EARH for real-time tasks, in which a rolling-horizon policy was used to enhance the system's schedulability. Additionally, the resource scaling up and resource scaling down strategies were developed and integrated into EARH, which can flexibly adjust the active hosts' scale so as to meet the tasks' real-time requirements and save energy.

The EARH algorithm is the first of its kind reported in the literature; it comprehensively addresses the issue of real-time, schedulability, elasticity, and energy saving. To evaluate the effectiveness of our EARH, we conduct extensive simulation experiments to compare it with other algorithms. The experimental results indicate that EARH can efficiently improve the scheduling quality of others in different workloads and is suitable for energy-aware scheduling in virtualized clouds.

The following issues will be addressed in our future studies. First, we will apply vertical scaling of VMs in terms of CPU in our energy-aware model, i.e., the maximum amount of CPU cycles assigned to a VM that runs a task can be updated dynamically. Second, we plan to implement the EARH in a real cloud environment to test its performance.

Acknowledgments

The authors are grateful to the anonymous referees for their insightful suggestions and comments. This research was supported by the National Natural Science Foundation of China under Grants No. 61104180, 91024030, the Southwest electron & telecom technology institute under Grant No. 2013001.

References

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the Fifth Utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599-616, 2009.

[2] A.V. Dastjerdi, S.G.H. Tabatabaei, and R. Buyya, "A Dependency-Aware Ontology-Based Approach for Deploying Service Level Agreement Monitoring Services in Cloud," *Software-Practice and Experience*, vol. 42, pp. 501-518, 2012.

[3] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-Aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing," *Future Generation Computer Systems*, vol. 28, pp. 755-768, 2012.

[4] X. Zhu and P. Lu, "Study of Scheduling for Processing Real-Time Communication Signals on Heterogeneous Clusters," *Proc. Ninth Int'l Symp. Parallel Architectures, Algorithms, and Networks (I-SPAN '08)*, pp. 121-126, May 2008.

[5] J.G. Koomey, "Estimating Total Power Consumption by Servers in the U.S. and the World,"report, Lawrence Berkeley Nat'l Laboratory, Stanford Univ., 2007.

[6] W.-C. Feng, "Making a Case for Efficient Supercomputing," *Queue*, vol. 1, no. 7, pp. 54-64, 2003.

[7] H. Cademartori, *Green Computing Beyond the Data Center*, http://www.powersavesoftware.com/Download/PS WP GreenComputing EN.pdf, 2007.

[8] L.A. Barroso and U. Hlzle, "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1-108, 2009.

[9] X. Fan, W.D. Weber, and L.A. Barroso, "Power Provisioning for a Warehouse-Sized Computer," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 13-23, 2007.

[10] Y.-K. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computation Survey*, vol. 31, no. 4, pp. 406-471, 1999.

[11] X. Qin and H. Jiang, "A Novel Fault-Tolerant Scheduling Algorithm for Precedence Constrained Tasks in Real-Time Heterogeneous Systems," *J. Parallel Computing*, vol. 32, no. 5, pp. 331-356, 2006.

[12] X. Zhu, C. He, K. Li, and X. Qin, "Adaptive Energy-Efficient Scheduling for Real-Time Tasks on DVS-Enabled Heterogeneous Clusters," *J. Parallel and Distributed Computing*, vol. 72, pp. 751-763, 2012.

[13] S. Zikos and H.D. Karatza, "Performance and Energy Aware Cluster-Level Scheduling of Compute-Intensive Jobs with Unknown Service Times," *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 239-250, 2011.

[14] J.S. Chase, D.C. Anderson, P.N. Thakar, A.M. Vahdat, and R.P. Doyle, "Managing Energy and Server Resources in Hosting Centers," *Proc. 18th ACM Symp. Operating Systems Principles (SOSP '01)*, pp. 103-116, Oct. 2001.

[15] R. Ge, X. Feng, and K.W. Cameron, "Performance-Constrained Distributed DVS Scheduling for Scientific Applications on Power-Aware Clusters," *Proc. ACM/IEEE Conf. Supercomputing (SC '05)*, p. 34, Nov. 2005.

[16] K.H. Kim, R. Buyya, and J. Kim, "Power-Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-Enabled Clusters," *Proc. Seventh IEEE Int'l Symp. Cluster Computing and the Grid (CCGrid '07)*, pp. 541-548, May 2007.

[17] V. Nélis, J. Goossens, R. Devillers, D. Milojevic, and N. Navet, "Power-Aware Real-Time Scheduling upon Identical Multiprocessor Platforms," *Proc. IEEE Int'l Conf. Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC '08)*, pp. 209-216, June 2008.

[18] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing,"Technical Report UCB/EECS-2009-28 Univ. of California, Berkeley, 2009.

[19] L. Liu, H. Wang, X. Liu, X. Jin, W. He, Q. Wang, and Y. Chen, "GreenCloud: A New Architecture for Green Data Center," *Proc. Sixth Int'l Conf. High Performance Distributed Computing (HPDC '08)*, pp. 29-38, June 2008.

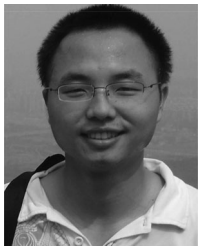
[20] V. Petrucci, O. Loques, and D. Mossé, "A Dynamic Configuration Model for Power-Efficient Virtualized Server Clusters," *Proc. 11th Brazillian Workshop Real-Time and Embedded Systems (WTR '09)*, May 2009.

[21] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic Provisioning Modeling for Virtualized Multi-Tier Applications in Cloud Data Center," *Proc. Third IEEE Int'l Conf. Autonomic Computing (ICAC '06)*, pp. 15-24, June 2006.

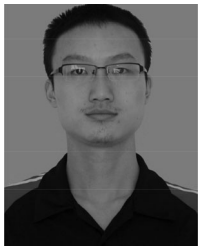
[22] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems," *Proc. Ninth ACM/IFIP/USENIX Int'l Conf. Middleware (Middleware '08)*, pp. 243-264, Dec. 2008.
[23] Í. Goiri, J.L. Berral, J.O. Fitó, F. Julià, R. Nou, J. Guitart, R. Gavaldà, and J. Torres, "Energy-Efficient and Multifaceted Resource Management for Profit-Driven Virtualized Data Centers," *Future Generation Computer Systems*, vol. 28, pp. 718-731, 2012.
[24] X. Wang, Z. Du, and Yi Chen, "An Adaptive Model-Free Resource and Power Management Approach for Multi-Tier Cloud Environments," *The J. Systems and Software*, vol. 85, pp. 1135-1146, 2012.
[25] P. Graubner, M. Schmidt, and B. Freisleben, "Energy-Efficient Management of Virtual Machines in Eucalyptus," *Proc. IEEE Fourth Int'l Conf. Cloud Computing (CLOUD '11)*, pp. 243-250, 2011.
[26] L. Yan, J. Luo, and N.K. Jha, "Joint Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-Time Embedded Systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 1030-1041, July 2005.
[27] A. Beloglazov and R. Buyya, "Optimal on 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, pp. 1397-1420, 2012.
[28] L. Wu, G. Kumar, and R. Buyya, "SLA-Based Admission Control for a Software-as-a-Service Provider in Cloud Computing Environments," *J. Computer and System Science*, vol. 78, no. 5, pp. 1280-1299, 2012.
[29] J.O. Gutierrez and K.M. Sim, "A Family of Heuristics for Agent-Based Elastic Cloud Bag-of-Tasks Concurrent Scheduling," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1682-1699, 2013.
[30] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F.D. Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23-50, 2011.
[31] *Google Cluster Data V2*, http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1, 2011.
[32] C. Reiss, J. Wilkes, and J. Hellerstein, "Google Cluster-Usage Traces: Format + Schema," White Paper, Google Inc., 2011.
[33] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Comm. of the ACM*, vol. 51, no. 1, pp. 107-113, 2008.
[34] I.S. Moreno, P. Garraghan, P. Towend, and X. Jie, "An Approach for Characterizing Workloads in Google Cloud to Derive Realistic Resource Utilization Models," *Proc. IEEE Seventh Int'l Symp. Service-Oriented System Eng. (SOSE '13)*, pp. 49-60, 2013.

Xiaomin Zhu received the BS and MS degrees in computer science from Liaoning Technical University, China, in 2001 and 2004, respectively, and the PhD degree in computer science from Fudan University, Shanghai, China, in 2009. In the same year, he received the Shanghai Excellent Graduate Award. He is currently an associate professor in the College of Information Systems and Management at the National University of Defense Technology, Changsha, China. His research interests include scheduling and resource management in green computing, cluster computing, cloud computing, and multiple satellites. He has published more than 50 research articles in refereed journals and conference proceedings such as *IEEE Transactions on Computers*, *IEEE Transactions on Parallel & Distributed Systems*, *Journal of Parallel and Distributed Computing*, *Journal of Systems and Software*, and so on. He is also a frequent reviewer for international research journals, e.g., *IEEE Transactions on Computers*, *IEEE Transactions on Network and Service Management*, *IEEE Transactions on Signal Processing*, *Journal of Parallel and Distributed Computing*, etc. He is a member of the IEEE, the IEEE Communication Society, and the ACM.

Laurence T. Yang received the BE degree in computer science and technology from Tsinghua University, China, and the PhD degree in computer science from the University of Victoria, Canada. He is currently a professor in the Department of Computer Science, St. Francis Xavier University, Canada. His research interests include parallel and distributed computing, embedded and ubiquitous/pervasive computing. His research has been supported by the National Sciences and Engineering Research Council, and the Canada Foundation for Innovation. He is a member of the IEEE.

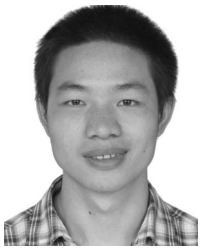


Huangke Chen received the BS. degree in information systems from the National University of Defense Technology, China, in 2008. He is currently working toward the MS degree in the College of Information System and Management at the National University of Defense Technology. His research interests include cloud computing and green computing.



Ji Wang received the BS degree in information systems from the National University of Defense Technology, China, in 2008. He is currently working toward the MS degree in the College of Information System and Management at the National University of Defense Technology. His research interests include real-time systems, fault-tolerance, and cloud computing.

Shu Yin received the PhD degree from the Department of Computer Science and Software Engineering at Auburn University in 2012. He is currently an assistant professor in the School of Information Science and Engineering at Hunan University, China. His research interests include storage systems, reliability modeling, fault tolerance, energy-efficient computing, and wireless communications. He is a member of the IEEE.



Xiaocheng Liu received the PhD degree from the National University of Defense Technology in 2012. He is currently an assistant professor in the College of Information Systems and Management at the National University of Defense Technology, Changsha, China. His recent research interests include resource allocation in the cloud, simulation-based training, and component- based modeling.