Energy-Aware Rolling-Horizon Scheduling for Real-Time Tasks in Virtualized Cloud Data Centers

Xiaomin Zhu, Huangke Chen
Science and Technology on Information
Systems Engineering Laboratory
National University of Defense Technology
Changsha 410073, China
Email: {xmzhu, hkchen}@nudt.edu.cn

Laurence T. Yang
Department of Computer Science
St. Francis Xavier University
Antigonish, NS, B2G 2W5, Canada
Email: ltyang@stfx.ca

Shu Yin
School of Information
Science and Engineering
Hunan University
Changsha 410012, China
Email: shuyin@hnu.edu.cn

Abstract—Developing energy-aware Cloud data centers not only can reduce power electricity cost but also can improve system reliability. Existing scheduling algorithms developed for energy-aware Cloud data centers commonly lack the consideration of task level scheduling. To address this issue, we propose a novel rolling-horizon scheduling architecture for real-time task scheduling. Besides, a task energy consumption model is given in detail. Based on the novel scheduling architecture, we develop a novel energy-aware scheduling algorithm EARH for real-time, aperiodic tasks. The EARH employs a rolling-horizon optimization policy and can be extended to integrate other scheduling algorithms. Again, we propose the resource scaling up and scaling down strategies to make a good trade-off between task’s schedulability and energy saving. Extensive experiments are conducted to validate the superiority of our EARH by comparing it with three baselines. Experimental results show that EARH significantly improves the scheduling quality of others and it is suitable for real-time task scheduling in virtualized Cloud data centers.

I. INTRODUCTION

The Cloud 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 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 service providers [2].

It is worthwhile to note that more and more large-scale data centers were built, which results in consuming tremendous amount of energy. It is estimated that data center servers currently consume about 0.5% of the total electricity consumption in the world, and the percentage will be quadruple by 2020 if the current demand goes on [3]. Besides, additional energy is needed for cooling equipments that control the temperature of data centers to maintain system reliability [4]. Furthermore, high energy consumption has negative impacts on environment [5]. Consequently, it is highly desirable to employ some measures to reduce energy consumption of Cloud data centers.

In the meantime, it is necessary and, in some cases, mandatory to provide some applications with real-time guarantee, in which the correctness depends not only on the logic results, but also on the time instants at which these results are produced, e.g., the real-time battle field information processed by a Cloud data center. Missing deadlines on getting the formation may yield catastrophic consequence. It is clear that real-time response in this scenario is more crucial than energy conservation. Hence, more machines should be active to finish these real-time tasks. Conversely, if some tasks are not emergent, the data center may use less machines to reduce energy consumption satisfying the timing needs of users.

Thanks to the development of virtualization technology, virtual machines (VMs) can be dynamically created, migrated, and canceled, which forms a key characteristic of virtualized Cloud elasticity. However, to the best of our knowledge, the current elasticity in Cloud is not well managed, which makes room for studying more efficient approaches to make a good trade-off between real-time guarantee and energy saving.

Motivated by the above arguments, we attempt to in this paper incorporate the elasticity and tasks’ real-time needs into energy-aware scheduling strategies, as well as design and implement a novel energy-aware scheduling strategy for independent tasks in a Cloud data center.

The major contributions of this paper are as follows:

1) We proposed an energy-aware scheduling scheme by rolling-horizon optimization, and we introduced a scheduling architecture for rolling-horizon optimization.

2) We developed some policies for VMs’ creation, migration and canceling to dynamically adjust the scale of Cloud, meeting the real-time requirements and striving to save energy.

3) We put forward an energy-aware rolling-horizon scheduling algorithm EARH for real-time tasks in a Cloud data center.

4) We simulated a Cloud data center where the EARH algorithm was implemented and evaluated.

The rest of this paper is organized as follows: In Section II, we summarize the related work. Section III gives the problem depictions. In Section IV, we describe the EARH algorithm and its main principles. In Section V, simulation experiments are carried out. Section VI concludes this paper.

II. RELATED WORK

Green computing and energy conservation in modern distributed computing context are receiving a great deal of
attention in the research community. Meanwhile, efficient scheduling methods in this issue have been overwhelmingly investigated. For example, Zhu et al. proposed the concept of slack sharing on multi-processor systems to reduce energy consumption and investigated two power-aware scheduling algorithms based on slack sharing for task sets with and without precedence constraints on multiprocessor systems [6]. Routree et al. studied a linear programming system using DVFS technique to deliver near-optimal schedules in which the allowable time delays, communication slack and memory pressure are considered to deal with energy conservation for a given pre-generated schedule [7]. Yu et al. proposed an off-line energy-aware allocation policy formulated as an extended generalized assignment problem [8]. Even though the aforementioned scheduling policies are able to achieve high scheduling quality in energy reduction, they are static ones not suitable for dynamic environment such as Cloud where the tasks’ arrival times are not known a priori.

There also exist a lot of dynamic scheduling algorithms to reduce energy consumption. For instance, 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 [9]. Zikos and Karatzis proposed a performance and energy-aware scheduling algorithms in cluster for compute-intensive jobs with unknown service time [10]. Gu et al. studied distributed performance-directed DVFS scheduling strategies that can make energy savings without increasing execution time by varying scheduling granularity [11]. It should be noted that these scheduling schemes do not consider the virtualization technique, which cannot efficiently improve the system resource utilization.

Virtualization technology has become an essential tool for providing resource flexibly to each user and isolating security and stability issues from other users [12]. Therefore, an increased number of data centers employ the virtualization technology when managing resources. Correspondingly, many energy-efficient scheduling algorithms for virtualized Cloud data centers were designed. For example, Liu et al. aimed to reduce energy consumption in virtualized data centers by supporting virtual machine (VM) migration and VM placement optimization while reducing the human intervention [13]. 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 [14]. Bi et al. suggested a dynamic resource provisioning technique for cluster-based virtualized multi-tier applications. In their approach, a hybrid queuing model was employed to determine the number of VMs at each tier [15]. Verna 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 [16]. Beloglazov et al. proposed some heuristics for dynamic adaption of VM allocation at run-time based on the current utilization of resources by applying live migration, switching idle nodes to sleep mode [17]. Goiri et al. presented an energy-efficient and multifacet 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 [18]. Wang et al. investigated adaptive model-free approaches for resource allocation and energy management under time-varying workloads and heterogeneous multi-tier applications and multiple metrics including throughput, rejection amount, queuing state were considered to design resource adjustment schemes [19]. Unfortunately, to the best of our knowledge, no work considers the dynamic energy-efficient scheduling issue for real-time tasks in virtualized Cloud data centers. 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 to save energy by dynamic VMs consolidation.

III. PROBLEM FORMULATION

A. Scheduling Model

We target a virtualized data center that is characterized by an infinite set $H = \{h_1, h_2, \cdots\}$ 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), 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 $k$th host, respectively. For each host $h_k$, it contains a set $V_k = \{v_{1k}, v_{2k}, \cdots, v_{jk}\}$ 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 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.

![Fig. 1. Scheduling architecture.](image-url)
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 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 tasks in waiting pool including their deadlines, currently allocated VMs, start time, etc.

**Step 2.** Sort the tasks in rolling-horizon by their deadlines to facilitate the scheduling operation.

**Step 3.** The real-time controller determines whether a task in rolling-horizon can be finished before its deadline. The VM controller adds VMs to finish the task within timing constraint if current VMs cannot finish it successfully. If no schedule can be found to satisfy the task’s timing requirement although enough VMs has been added by testing, the task will be rejected. Or the task will be retained in the rolling-horizon.

**Step 4.** Update the scheduling decision for the tasks in rolling-horizon, e.g., their execution order, start time, allocated VMs and new active hosts.

**Step 5.** When a task in the rolling-horizon is ready to execute, dispatch the task 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 task information, and then decides whether some VMs should be stopped or migrated to consolidate resources so as to save energy.

### B. 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 $t_i$. We let $dt_{ijk}$ 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 $v_{jk}$.

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

(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:

$$ft_{ijk} = st_{ijk} + et_{ijk}.$$  

(2)

In addition, $x_{ijk}$ is employed to reflect a mapping of tasks to VMs at different hosts in a virtualized Cloud data center, 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.,

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

(3)

### C. 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 [17]. Further, the energy consumption can be classified into two parts, i.e., dynamic energy consumption and static (i.e., leakage) energy consumption [20]. 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 $e_{cijk}$ 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 $ec^r_{ijk}$ and the energy consumption $ec_{ijk}$ can be calculated as follows:

$$ec_{ijk} = ec^r_{ijk} \cdot et_{ijk}.$$  

(4)

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

$$e_{c\text{exec}} = \sum_{k=1}^{[H_a]} \sum_{j=1}^{[V_{ik}]} \sum_{t=1}^{[T]} x_{ijk} \cdot ec_{ijk} = \sum_{k=1}^{[H_a]} \sum_{j=1}^{[V_{ik}]} \sum_{t=1}^{[T]} x_{ijk} \cdot ec^r_{ijk} \cdot et_{ijk}.$$  

(5)

We assume in (5) that no energy consumption is incurred when VMs are sitting idle. However, this assumption is not valid in real-world virtualized Cloud data centers. This energy consumption when VMs are idle includes two parts, i.e., all the VMs in a host are idle and some of VMs 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 $ec^s_{jk}$. Suppose the idle time when all the VMs in a host $h_k$ are idle is $i_t$, the energy consumption when a host is idle (i.e., all the VMs in this host are idle) can be written as:

$$e_{c\text{idle}} = \sum_{k=1}^{[H_a]} \sum_{j=1}^{[V_{ik}]} ec^s_{jk} \cdot i_t.$$  

(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 $ec^r_{jk}$. Then, we obtain the analytical formula for the energy consumed in this case as:

$$e_{c\text{partidle}} = \sum_{k=1}^{[H_a]} \sum_{j=1}^{[V_{ik}]} ec^r_{jk} \cdot ec_{partidle} = \sum_{k=1}^{[H_a]} \sum_{j=1}^{[V_{ik}]} \left( \max_{t=1}^{[T]} (f_{t} - d_{t}) - \sum_{i=1}^{[T]} x_{ijk} \cdot et_{ijk} \right).$$  

(7)
where $\tau_j^{idle}$ 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:

$$e_{cei} = e_c^{exc} + e_c^{idle} + e_c^{partIdle}$$

$$= \sum_{k=1}^{[H_k]} \sum_{j=1}^{[V_k]} [T] \cdot x_{ijk} \cdot e_{crj;k} \cdot t_{i,jk}$$

$$+ \sum_{k=1}^{[H_k]} \sum_{j=1}^{[V_k]} e_{crj;k} \cdot t_{ik}$$

$$+ \sum_{k=1}^{[H_k]} \sum_{j=1}^{[V_k]} [T] \cdot \max_{j=1}^{[V_k]} \{ f_j \} \cdot t_{i,k} - \sum_{j=1}^{[V_k]} x_{ijk} \cdot e_{t,ijk}, \quad (8)$$

Noticeably, perhaps the host is not fully utilized, e.g., although some VMs are placed on a host, maybe some resource is still unused. However, the resource also consume energy. Suppose there are $s$ periods in 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.

$$e_{cur} = \sum_{k=1}^{[H_k]} \sum_{p=1}^{s} \left( e_{crj} \cdot \frac{[V_k(p)]}{\sum_{j=1}^{[V_k]} e_{crj}} \right) \cdot t_p, \quad (9)$$

where $[V_k(p)]$ denotes the VMs’ count in $p$th period of host $h_k$, and $e_{crj}$ represents the energy consumption rate 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:

$$e_{c} = e_{cei} + e_{cur}. \quad (10)$$

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 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 data center. Our EARTH 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.

IV. THE EARTH SCHEDULING STRATEGY

A. 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 (RH) and their schedules are allowed to be adjusted for the schedulability of the new task and possibly less energy consumption. One essential advantage of RH optimization is that the task migration required by rescheduling does not yield any overhead because all the tasks are waiting in the rolling-horizon. 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: \hspace*{0.2cm} $Q \leftarrow NULL; \ R \leftarrow NULL$;
3: \hspace*{0.2cm} Add a new task $t_i$ into set $Q$;
4: \hspace*{0.2cm} for each task $t_o$ do
5: \hspace*{0.4cm} if $st_{i,jk} > a_i$ and $x_{i,jk} == 1$ then
6: \hspace*{0.6cm} Add task $t_o$ into set $Q$;
7: \hspace*{0.4cm} end if
8: \hspace*{0.4cm} if $st_{i,jk} + ct_{i,jk} \geq rt_{i,jk}$ and $x_{i,jk} == 1$ then
9: \hspace*{0.6cm} $rt_{i,jk} \leftarrow st_{i,jk} + ct_{i,jk}$;
10: \hspace*{0.6cm} Update the ready time of $v_{jk}$ in Vector $R$;
11: \hspace*{0.4cm} end if
12: \hspace*{0.4cm} Sort tasks in $Q$ by their deadlines in a non-descending order;
13: \hspace*{0.4cm} for each task $t_o$ in set $Q$ do
14: \hspace*{0.6cm} Schedule task $t_o$ by Different Energy-Efficient Scheduling Algorithms;
15: \hspace*{0.6cm} if $x_{i,jk} == 0$ then
16: \hspace*{0.8cm} Reject task $t_o$;
17: \hspace*{0.6cm} end if
18: \hspace*{0.6cm} end for
19: \hspace*{0.6cm} Update scheduling decisions;
20: \hspace*{0.4cm} end for
21: end for

B. Energy-Aware Scheduling Algorithm

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

$$st_{i,jk} = \max\{rt_{i,jk}, a_i\}, \quad (11)$$

where $rt_{i,jk}$ 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 $rt_{i,jk}$ of $v_{jk}$ is:

$$rt_{i,jk} = st_{i,jk} + ct_{i,jk}. \quad (12)$$

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

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:  

1. **Step 1.** Create a new VM in a current active host without any VM migration;  
2. **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 new VM on it;  
3. **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 [21]:

$$mt(v_{jk}) = \frac{r(v_{jk})}{n(v_{jk})}, \quad (13)$$
Algorithm 2: Pseudocode of energy-efficient scheduling

1: for each task \( t_i \) in set \( Q \) do
2: \( \text{findTag} \leftarrow \text{FALSE}; \text{findVM} \leftarrow \text{NULL}; \)
3: for each VM \( v_{ijk} \) in the system do
4: Calculate the start time \( st_{ijk} \) by Eq. (11) and the execution time \( et_{ijk} \) by Eq. (1);
5: if \( st_{ijk} + et_{ijk} \leq d_i \) then
6: \( \text{findTag} \leftarrow \text{TRUE}; \)
7: Calculate \( cc_{ijk} \) by Eq. (4);
8: end if
9: end for
10: if \( \text{findTag} == \text{FALSE} \) then
11: \( \text{scaleUpResource}(); \)
12: end if
13: if \( \text{findTag} == \text{TRUE} \) then
14: Select \( v_{ijk} \) with minimal energy consumption to execute \( t_i \);
15: \( \text{findVM} \leftarrow v_{ijk}; \)
16: end if
17: end for
18: Update the scheduling decision of \( t_i \) and remove it from \( Q \);
19: end for

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

\[
st_{ijk} = \begin{cases} 
    a_i + cl(v_{ijk}), & \text{if setp1,} \\
    a_i + cl(v_{ijk}) + \sum_{p=1}^{l_{ijk}} mt(v_{ijk}), & \text{if setp2,} \\
    a_i + st(h_k) + cl(v_{ijk}), & \text{if setp3.}
\end{cases}
\] (14)

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

When 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 reduce the total energy consumption by the data center. Algorithm 4 gives the pseudocode of algorithm about scaling down resources.

V. PERFORMANCE EVALUATION

To demonstrate the performance improvements gained by EARTH, we quantitatively compare it with three baseline algorithms - non-RH-EARTH (NRHEARTH in short), non-Migration-EARTH (NMEARTH in short), and non-RH-Migration-EARTH (NRHMREARTH in short). The three baseline algorithms are briefly described as follows:

NRHEARTH: Differing from EARTH, NRHEARTH does not employ the rolling-horizon optimization;

NMEARTH: Differing from EARTH, NMEARTH does not employ VM migration while allocating real-time tasks;

NRHMREARTH: Differing from EARTH, NRHMREARTH does not employ rolling-horizon optimization and VM migration.

The performance metrics by which we evaluate the system performance include:
1) Guarantee Ratio (GR) defined as: \( GR = \text{Total count of tasks guaranteed to meet their deadlines/Total number of tasks} \);
2) Total energy consumption (TEC) is: \( \text{Total energy consumed by hosts} \).

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 \( v_{ijk} \); \( \text{findTag} \leftarrow \text{TRUE}; \) break;
6: end if
7: end for
8: if \( \text{findTag} == \text{FALSE} \) then
9: Search the host \( h_a \) with minimal CPU utilization;
10: Find the VM \( v_{pa} \) with minimal MIPS in \( h_a \);
11: for each host \( h_k \) except \( h_a \) in \( H_a \) do
12: if VM \( v_{pa} \) can be added in host \( h_k \) then
13: Migrate VM \( v_{pa} \) to host \( h_k \); break;
14: end if
15: end for
16: if \( \text{findTag} \) can be added in host \( h_a \) then
17: Create VM \( v_{pa} \);
18: if \( t_i \) can be finished in \( v_{pa} \) before its deadline then
19: \( \text{findTag} \leftarrow \text{TRUE}; \)
20: end if
21: end if
22: end if
23: if \( \text{findTag} == \text{FALSE} \) then
24: Start a host \( h_n \) and put it in \( H_a \);
25: Create VM \( v_{pn} \) on \( h_n \);
26: if \( t_i \) can be finished in \( v_{pn} \) before its deadline then
27: \( \text{findTag} \leftarrow \text{TRUE}; \)
28: end if
29: end if

3) Energy Per Task (EPT) calculated as: \( \text{EPT} = \frac{\text{Total energy consumption}}{\text{Accepted task count}} \).

A. Simulation Method and Parameters

In order to ensure the repeatability of experiments, we chose the way of simulations to evaluate the performance of aforementioned algorithms. In our simulations, the CloudSim toolkit [22] has been chosen as a simulation platform, and also we add some new settings to finish 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 can be 1000 MIPS, 1500 MIPS, or 2000 MIPS;
2) The powers of the three different CPU performances are 250W, 200W, or 400W;
3) The start-up time of a host is 90s and the creation time of a VM is 15s;
4) Parameter taskCount is used to represent the task count in our simulations;
5) We employ baseDeadline to control the task deadline which can be calculated as:
\[
d_i = a_i + \text{baseDeadline},
\] (15)
where parameter baseDeadline is in uniform distribution \( U(\text{baseTime}, a \times \text{baseTime}) \) and we set \( a = 4 \);
6) The arrived task count in a time unit is Poisson distribution, and parameter intervalTime is used to denote the time interval between two consecutive tasks.
Algorithm 4 Pseudocode of Scaling Down Algorithm

1: $SH \leftarrow \emptyset; \quad DH \leftarrow \emptyset$
2: for each VM $v_{jk}$ in the system do
3: if $v_{jk}$'s idle time $i_{jk} > THRESH$ 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 $H_s$ do
8: if there is no VM on $h_k$ then
9: shut down host $h_k$ and remove it from $H_s$;
10: end if
11: end for
12: sort the hosts in $H_s$ in an increasing order of the CPU utilization;
13: $SH \leftarrow H_s; \quad DH \leftarrow H_s$ 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; $SH \leftarrow SH - AH - h_k; \quad DH \leftarrow DH - h_k$;
29: shut down host $h_k$ and remove it from $H_s$;
30: end if
31: end for

The values of parameters are listed in Table I.

B. Performance Impact of Task Count

In this section, we present a group of experimental results to observe the performance of EARH, NRH- EARH, NMEARH, and NRHMEARH. Fig. 2 shows the experimental results.

We can observe from Fig. 2(a) 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 trend 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 timing constraint. Besides, it can be found that EARH and NMEARH have higher guarantee ratios than another two algorithms. This can be explained that EARH and NMEARH employ the rolling-horizon optimization policy that is able to make those tasks with tight deadlines finish earlier, so the schedulability is efficiently improved.

From Fig. 2(b), it can be observed that although NRH- EARH conserves the most energy, its guarantee ratio is the worst (See Fig 2(a)); in contrast, NMEARH has the most energy consumption but with higher guarantee ratio, which reflects that NRH- EARH and NMEARH lack of good trade-off between guarantee ratio and total energy consumption. Moreover, we can observe that EARH and NRHMEARH 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 on it, which avoids the energy consumption by adding new active host. 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. 2(c) reveals that the EPTs of NMEARH and NRHMEARH slightly increase with the increase of task count.
whereas other two algorithms that employ VM migration policy basically maintain stable EPT. This experimental result can be explained that when the task count increases, new VMs are needed to accommodate these real-time tasks. For NMEARH and NRHMEARH, they do not employ the VM migration policy and thus have to start some hosts and create VMs on them if no enough idle resource is available, which definitely increases the energy consumption per task. However, RAHEH and NRHHEARH use the migration policy striving to avoid starting new hosts, and therefore the current resource is efficiently utilized leading to a basically stable value of EPT.

C. Performance Impact of Task Arrival Rate

In order to examine the performance impact of task arrival rate, we vary the value of intervalTime from 1 to 11 with step 2. Fig. 3 illustrates the performance of the four algorithms.

The observations from Fig. 3(a) show that the four algorithms mainly have unchanged guarantee ratios no matter how the task count varies. This comes 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 needed to accommodate these tasks. In addition, EARTH and NMEARH with rolling-horizon optimization have higher guarantee ratio than NRHEARH and NRHHEARH without using rolling-horizon. The explanation for this experimental result is like that in Fig. 2(a).

Fig. 3(b) shows that NMEARH and NRHMEARH consume more energy than EARTH and NRHEARH. This reason can be explained as that in Fig. 2(b). Besides, we can get from Fig. 3(b) that when the parameter intervalTime changes, the energy consumption by the four algorithms is basically unchanged indicating that the task arrival rate has little impact on energy consumption.

The experimental results from Fig. 3(c) depict that when intervalTime changes, the EPTs of the four algorithms are basically constant demonstrating that task arrival rate has little impact on EPT. Further, the EPTs of EARTH and NMEARH are obviously less than that of NMEARH and NRHMEARH with the explanation like that in Fig. 2(c).

D. Performance Impact of Task Deadline

It is observed from Fig. 4(a) that with the increase of baseTime (i.e., task deadline becomes looser), the guarantee ratios of the four algorithms increase. This is because the deadlines are prolonged making tasks can be finished later within timing constraint. In addition, Fig. 4(a) shows that NMEARH and NRHEARH have the highest and lowest guarantee ratios, respectively. This can be explained that after employing the rolling-horizon policy, the tasks with tight deadlines can be preferentially finished and those task with loose deadlines
can be delayed to finish. Thus, the guarantee ratio enhances. Again, when the deadline is tight, NMEARH starts more hosts because it does not use VM migration policy, which makes the tasks arrive later finish successfully due to more active available resource. Therefore, its guarantee ratio is a little bit higher than that of EARH and NRHMEARH which do employ the VM migration policy.

From Fig. 4(b), we can see that when $baseTime$ increases, the energy consumption by all the algorithms increases correspondingly. This is because when the deadline becomes looser, more tasks can be finished within their deadlines and so more energy is consumed. Also, the energy consumptions by EARH and NRHMEARH are less than that of others and the trend becomes more pronounced with the increase of $baseTime$. This is because EARH and NRHMEARH use the VM migration policy that can efficiently utilize the resource of active hosts, which avoids starting more hosts to finish tasks. However, NMEARH and NRHMEARH can only constantly start hosts to finish more tasks resulting in more energy consumption.

Fig. 4(c) shows that the EPT's of NMEARH and NRHMEARH become larger with the increase of $baseTime$. This can be explained that NMEARH and NRHMEARH start more hosts and thus yield more idle resource resulting in lower utilization. When the value of $baseTime$ is less than 300, the EPT's of EARH and NRHMEARH decrease with the explanation that when the deadlines become looser, more tasks can be finished in the current active hosts. Again, the VM migration policy is employed without starting hosts. Hence, the utilization of active hosts is higher. Nevertheless, when the value of $baseTime$ is larger than 300, the current active hosts lack the ability to finish more accepted tasks due to loose deadline and some hosts must be started, yielding some idle resource. Therefore, the EPT's increase correspondingly.

VI. CONCLUSIONS

In this paper, we proposed a rolling-horizon scheduling architecture and energy consumption model in a virtualized Cloud data center. Besides, based on the scheduling architecture, we presented a novel energy-aware scheduling algorithm EARH for aperiodic, independent real-time tasks. The EARH employs a rolling-horizon optimization policy to enhance the system schedulability. In addition, the resource scaling up, and resource scaling down strategies are 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. To validate the effectiveness of our EARH, we conduct extensive simulations. The experimental results show that EARH is better than other baselines in different workloads.

REFERENCES