STARS: Startup-Time-Aware Resource Provisioning and Real-Time Task Scheduling in Clouds

Xiaomin Zhu, Huangke Chen, Guipeng Liu
College of Information Systems and Management
National University of Defense Technology
Changsha, 410073, China
Email: {xmzhu, hkchen, laurelgp}@nudt.edu.cn

Ling Liu
College of Computing
Georgia Institute of Technology
266 Ferst Drive, Atlanta, GA 30332-0765
Email: lingliu@cc.gatech.edu

***Abstract*—Green cloud computing has become a major performance measure for many infrastructure as a service (IaaS) cloud data centers. A popular way to reduce energy consumption for virtualized clouds is to dynamically consolidate virtual machines (VMs) and turning off as many idle hosts as possible. Upon the increase of the system's workloads, the closed hosts will be restarted to meet the scale-up demand of resources. However, the time overhead of starting hosts and deploying VMs can delay the start time of real-time tasks, and may cause deadline violation of some real-time tasks. This problem can be further aggravated for heterogeneous physical hosts. In this paper, we propose a novel scheduling architecture that allocates an idle lash-up VM on each active host. Besides, we develop a startup-time-aware scheduling strategy to scale up the resource provisioning for these lash-up VMs to mitigate the performance impact of host machine startup time on real-time tasks. Furthermore, we propose a startup-time-aware scheduling algorithm, named *STARS*, striving to guarantee deadlines of real-time tasks, while exploiting the optimal operating frequencies and energy efficiencies of heterogeneous hosts to achieve energy conservation. We conduct extensive experiments to validate the efficiency of *STARS* using Googles workload traces. The experimental results show that *STARS* outperforms the existing scheduling algorithms in terms of guarantee ratio (up to 19.10%), energy saving (up to 29.40%) and resource utilization (up to 46.69%).**

***Index Terms*—cloud computing, virtualization, scheduling, real-time tasks, energy saving, startup time**

I. Introduction

Cloud computing has become a popular distributed computing paradigm for delivering on-demand services to customers in a "pay-as-you-go" cost model [1]. In order to satisfy the soaring demand of cloud computing services, many IT companies (e.g., Amazon, Google and IBM) are deploying geographically distributed data centers across different administrative domains around the world. Inevitably, the massive host (servers) in cloud data centers consume enormous amount of energy for computing and equipment cooling operations. It is reported that the energy consumed by data centers worldwide is about 1.5% of the global electricity use in 2010, and with the current growth trend, the percentage will double by 2020 [2]. Since the high energy consumption incurs tremendous energy-related costs, low reliability of physical hosts, and a large amount of CO_2 emissions [3], reducing energy consumption have become a major demand in virtualized cloud data centers.

One effective way for energy saving is to dynamically consolidate virtual machines (VMs) to a minimal number of physical hosts, and enable more idle hosts to be turned off [4], [5], [6]. There are two main reasons for energy saving by VM consolidation. First, with the development of virtualization technology, multiple VMs running on a single host can support multiple applications simultaneously [5], [7] with desired performance isolation, and this capability can also be utilized for reducing the total number of active physical hosts maintained in a cloud data center. It is reported that physical hosts in a completely idle state still dissipate over 50% as much power as when they are fully utilized [6], [8]. Thus, reducing active host count becomes an attractive solution for significant energy saving. Second, VMs can be migrated from one host to another without stopping applications running on them. When the overall workload on a server host decreases, and some of the VMs are in idle state, one can consolidate the active VMs to a small number of hosts by VM migration. This allows further energy saving by turning off larger number of under-utilized or idle hosts [6], [8].

However, VM consolidation requires careful planning for real-time applications with execution timing constraints. Today, many applications in the clouds are running real-time tasks with deadlines [4], [7]. These real-time tasks have two distinguishing features: they are submitted dynamically, and they demand both logical and temporal correctness of computations [5]. Although consolidating resources can effectively reduce the energy consumption for virtualized data centers, cloud platforms that support real-time applications are confronted with another challenging issue, i.e., when a large number of tasks arrives within very short time, the systems workload increases suddenly, the cloud platform usually responds to such workload burst by adding more hosts and deploying more VMs to meet the demand for more computing resources. However, the time overhead of starting new hosts and deploying VMs (about one minute) is non-negligible, which will further delay the start time of real-time tasks running on these host machines, and cause violation of deadlines of some tasks.

When multiple VMs simultaneously run on the same host, VMs access CPU resource in proportion to the weights that VMs have been assigned. For instance, a VM with a weight of 200 will get twice as much CPU as a VM with a weight of 100 on the same host. Furthermore, the weights of VMs can be recalculated and re-allocated at runtime [9]. Thus, virtualization technology has enabled resizing the CPU capacity of VMs dynamically [9], [10], [11]. In this paper, we leverage this technology to develop a real-time task scheduling scheme that offers startup-time-aware resource provisioning for clouds. First, we study how to mitigate the impact of host machine startup time on the timing constraints and deadline requirements of real-time tasks. Second, we investigate the optimal operation frequencies and energy efficiencies of heterogeneous hosts to achieve energy saving in data centers.

This paper makes three contributions: (1) We propose a startup-time-aware architecture for scheduling real-time tasks in virtualized clouds. (2) We develop a startup-time-aware strategy to mitigate the impact of host machine startup time on timing requirements of real-time tasks with respect to execution sequence and deadlines. (3) We design and implement a start-time aware real-time task scheduling algorithm *STARS*, to guarantee timing requirements of real-time tasks, while exploiting the optimal operating frequencies and energy efficiencies of heterogeneous VM hosts to save energy.

The rest of this paper is organized as follows. Section II briefly reviews the related work. Section III gives an overview of scheduling architecture and problem formulation, followed by the scheduling algorithm for real-time tasks in Section IV. In section V, we conduct experiments to evaluate the performance of our algorithm. Section VI concludes this paper.

II. Related Work

In recent years, the issue of high energy consumption in cloud data centers has attracted a great deal of attention. In response to that, a large amount of energy-efficient scheduling algorithms have been developed. Among them, there are three typical categories: (1) virtualization based, (2) DVFS (dynamic voltage and frequency scaling) based, and (3) combination of the two approaches above.

Virtualization-based energy-aware scheduling approaches appear in large numbers over the past several years. For instance, Beloglazov et al. proposed three heuristics to consolidate VMs dynamically, and then turn off idle hosts to reduce energy consumption [6]. Zhu et al. combined the rolling-horizon optimization technique and two resource scale strategies to form a novel scheduling algorithm EARH to trade-off tasks' schedulability and energy saving [4]. Quan et al. developed a reconfiguration algorithm, based on request prediction, to allocate hosts and VMs dynamically according to system's varied workload [12]. Wu proposed an algorithm to calculate optimal speed for task executions according to blocking conditions so as to minimize energy consumption [13]. Hsu et al. developed an energy-aware approach for virtual clusters by consolidating tasks [14]. Unfortunately, these approaches do not consider the impact of startup time of closed hosts on the timing requirements of real-time tasks, inevitably violating deadlines of some real-time tasks.

Another branch is DVFS-based methods. For example, Garg et al. developed near-optimal energy-efficient scheduling policies that leverage DVFS technique to minimize energy consumption and carbon emission and maximize the profit of the cloud providers [15]. Kim et al. provided several schemes for energy-aware provisioning of VMs for real-time services [7]. Ren et al. proposed an algorithm AGENT to make trade-offs among response time, resource utilization and energy consumption by leveraging DVFS technique [16]. Zhang et al. suggested an energy-aware algorithm called CloudFreq to schedule bag-of-tasks in DVFS-enabled clouds, making balance between makespan and energy saving [17]. Kamga et al. reported some experiments on dynamically scaling processor frequency according to VM CPU load to save energy [18]. However, only the dynamic power (less than 50% of host power [8]) can be controlled by DFVS technique, which has limited efficiency on energy saving of cloud data centers.

There also exist some work investigating the combination of VM consolidation and DVFS technique to save energy. For instance, Ding et al. developed a new VM scheduling algorithm, leveraging DVFS, to improve energy efficiency while executing tasks with deadlines in cloud data centers [19]. Lago et al. presented a scheduling algorithm to minimize the energy consumption in a cloud computing environment by using DVFS, load migration, and shutdown of under-utilized hosts [20]. Hanumaiah et al. combined DVFS, task migration and fan speed scaling techniques to make trade-off between system performance and energy conservation [21]. However, the aforementioned approaches have no capability to avoid the impact of machine startup overheads on the timing requirements of real-time tasks.

III. Modeling and problem formulation

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

A. Scheduling architecture

In this paper, we design a startup-time-aware scheduling architecture for virtualized clouds, as shown in Fig. 1. The architecture consists of three layers: user layer, scheduling layer and resource layer. In addition, the resource layer can be further divided into two layers: VM layer and host layer.

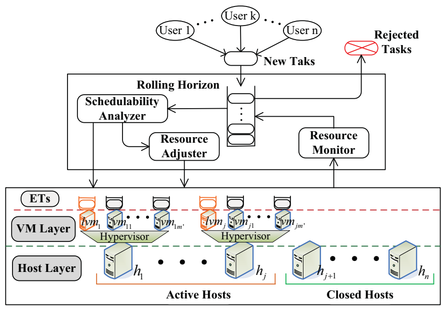

Fig. 1. The startup-time-aware scheduling architecture

In cloud environment, users dynamically submit their applications to cloud providers. For the applications, we focus on real-time, independent tasks denoted as $T = \{t_1, t_2, \cdots, t_m\}$. For a given task t_i, it can be modeled by $t_i = \{a_i, q_i, d_i\}$, where a_i, q_i and d_i represent the arrival time, computation length (in MHz), and deadline of task t_i, respectively.

We target a virtualized cloud that consists of n physical hosts $H = \{h_1, h_2, \cdots, h_n\}$. Each host is characterized by $h_j = \{(f_j^d, v_j^d), m_j, VM_j\}$ where $(f_j^d, v_j^d) = \{(f_j^1, v_j^1), (f_j^2, v_j^2), \cdots, (f_j^{max}, v_j^{max})\}$ is a discrete set of frequency-voltage pairs of host h_j, m_j is the memory size; and $VM_j = \{lvm_j\} \bigcup \{vm_{j,1}, vm_{j,2}, \ldots, vm_{j,|VM_j|-1}\}$ is the VM set on the host h_j in which lvm_j denotes the only one lash-up VM on host h_j; $vm_{j,k} \in VM_j, 1 \leq k \leq |VM_j| - 1$ represents the k-th VM on host h_j; and $|VM_j|$ is the count of VMs on host h_j. A lash-up VM can be modeled as $lvm_j = \{c(lvm_j), m(lvm_j)\}$ where $c(lvm_j)$ and $m(lvm_j)$ represent the CPU performance (in Hz) and memory required for VM lvm_j. Similarly, a non-lash-up VM is modeled as $vm_{j,k} = \{f_{j,k}, m_{j,k}\}$ where $f_{j,k}$ and $m_{j,k}$ are respectively the CPU performance (in Hz) and memory required for vm_{jk}.

As illustrated in Fig. 1, the scheduling layer consists of a rolling horizon (RH in short), a schedulability analyzer, a resource adjuster and a resource monitor. The RH holds both new tasks and waiting tasks, while the local queues of VMs only contain the executing tasks (ETs). The schedulability analyzer is responsible for generating task-to-VM mappings and plan of scaling the resources according to certain objectives and the resource information from resource monitor. The resource adjuster conducts the plan of scaling resources.

The advantages of this scheduling architecture are: (1) It can significantly improve the schedulability of real-time tasks. Since all the waiting tasks are on RH rather than on VMs directly, when new tasks with tight deadlines arrive, both the waiting tasks and new tasks will be rescheduled, which enables more tasks to be finished before their deadlines. (2) The lash-up VMs on active hosts are helpful for mitigating the impact of time overheads of starting hosts and deploying VMs on deadlines of real-time tasks, without starting more hosts and wasting unnecessary static energy consumption. When the system's workload increases, the lash-up VMs can be scaled up immediately to execute some tasks with tight deadlines to avoid the impact of time overhead of scaling resources on the start of tasks. In addition, these lash-up VMs are deployed on active hosts without starting more closed hosts, which is good for reducing hosts' idle energy consumption.

B. Energy model of hosts

In this paper, we consider the energy consumption of a physical host based on the CPU power. The power consumption of a CPU can be categorized into the power when it is idle and the power when it is active. For a CPU of host h_j, its active power consumption, p_j^{active}, can be described as:.

$$p_j^{active} \propto {v_j}^2 \cdot f_j. \tag{1}$$

where $(f_j, v_j) \in (f_j^d, v_j^d)$ is a frequency-voltage pair of h_j.

The dynamic power (p_j^{active}) of h_j can be approximated as:

$$p_j^{active} \propto {f_j}^3. \tag{2}$$

Let s_j be the section of power consumption by the idle host (e.g., 50% and 60%) and p_j^{max} be the maximum power consumption when host h_j is fully utilized. The power of host h_j can be written as follows [15]:

$$\begin{aligned} p_j &= p_j^{idle} + p_j^{active} \\ &= s_j \cdot p_j^{max} \cdot c_j^t + \frac{(1 - s_j) \cdot p_j^{max}}{(f_j^{max})^3} \cdot (f_j)^3, \end{aligned} \tag{3}$$

where $c_j^t \in \{1, 0\}$ indicates whether h_j is active at time instant t, it is 1 when h_j is active, and is 0, otherwise.

The total energy consumption (tec_j) by host h_j from time st to time et can be approximated as:

$$tec_j = \int_{st}^{et} (s_j \cdot p_j^{max} \cdot c_j^t + \frac{(1 - s_j) \cdot p_j^{max}}{(f_j^{max})^3} \cdot (f_j)^3) dt. \tag{4}$$

C. Problem formulations

Due to the heterogeneous processing capabilities of VMs, the parameters $et_{i,j,k}$, $st_{i,j,k}$ and $ft_{i,j,k}$ are used to represent the execution time, start time and finish time of task t_i on non-lash-up VM $vm_{j,k}$ respectively, and $et_{i,j,k} = q_i / f_{j,k}$ [4]. Obviously, the relation of the three parameters above is:

$$ft_{i,j,k} = st_{i,j,k} + et_{i,j,k}. \tag{5}$$

Similarly, the parameters $et_{i,j}$, $st_{i,j}$ and $ft_{i,j}$ denote the execution time, start time and finish time of task t_i on lash-up VM vm_j, and $et_{i,j} = q_i / c(lvm_j)$.

The status variable $x_{i,j,k}$ is used to record whether the deadline of task t_i on non-lash-up VM $vm_{j,k}$ is guaranteed or not. $x_{i,j,k}$ is 1 if task t_i is mapped to $vm_{j,k}$ and $ft_{i,j,k}$ is not larger than d_i, otherwise, $x_{i,j,k}$ equals 0, i.e.,

$$x_{i,j,k} = \begin{cases} 1, & \text{if } t_i \text{ is mapped to } vm_{j,k} \cap ft_{i,j,k} \leq d_i, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Similarly, the status variable $x_{i,j}$ is used to record whether the deadline of task t_i on lash-up VM lvm_j is guaranteed or not.

In this paper, the primary objective is to maximize the ratio of tasks finished before their deadlines, i.e.,

$$\text{Max} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{|VM_j|-1} \frac{x_{i,j,k}}{|T|} + \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{x_{i,j}}{|T|}. \tag{7}$$

The amount of resources required for VMs on a host must not be larger than the capacity of the host. This requirement forms the scheduling constraint as:

$$\begin{aligned} &f_j^{max} - \sum_{k=1}^{|VM_j|-1} f_{j,k} - c(lvm_j) \geq 0, \forall h_k \in H; \\ &m_j - \sum_{k=1}^{|VM_j|-1} m_{j,k} - m(lvm_j) \geq 0, \forall h_k \in H. \end{aligned} \tag{8}$$

Subject to constraints in (8), the secondary optimization objective is to reduce the total energy consumption (TEC)

for executing a set of tasks T from time instant st to et, i.e.,

$$\text{Min} \sum_{j=1}^{n} tec_j = \sum_{j=1}^{n} \int_{st}^{et} (s_j \cdot p_j^{max} \cdot c_j^t + \frac{(1-s_j) \cdot p_j^{max}}{(f_j^{max})^3} \cdot (f_j)^3) dt. \tag{9}$$

We also focus on maximizing the average resource utilization of hosts, so we have:

$$\text{Max} \sum_{i=1}^{m} \sum_{j=1}^{n} (\sum_{k=1}^{|VM_j|-1} x_{i,j,k} + x_{i,j}) \cdot q_i / (\sum_{j=1}^{n} f_j^{max} \cdot wt_j), \tag{10}$$

where wt_j represents the active time of host h_j.

IV. Algorithm Design

In this section, we design the algorithm *STARS* for resource provisioning and real-time task scheduling.

A. The strategy for scaling up lash-up VMs

We firstly define the urgent tasks as those real-time tasks, whose timing constraints cannot be guaranteed by initiated VMs and will be violated by the time overhead of adding new resources. In order to mitigate the impact of resources' startup time on the timing requirements of urgent tasks, we leverage the mechanism of resizing CPU dynamically to develop a three-step strategy to scale up lash-up VMs to execute these urgent tasks as follows.

Step 1. If the remaining CPU resource of an active host can satisfy the requirement of a lash-up VM, scale up the lash-up VM's CPU frequency directly.

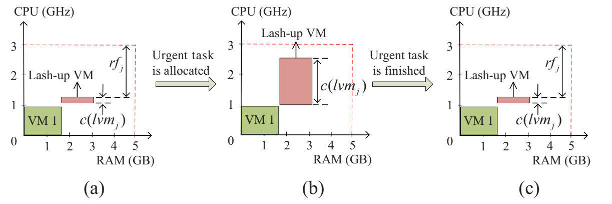

Fig. 2. An example of Step 1

We illustrate how the Step 1 works using an example in Fig. 2 where the horizontal axis and vertical axis respectively represent the memory and CPU resource of a host, and the red dashed lines indicate the host's maximal memory size and CPU frequency. Each rectangular represents a VM, and the length and height of a rectangular respectively refer to memory size and CPU frequency of a VM. In addition, we use $c(lvm_j)$ and rf_j to denote the CPU frequency of lash-up VM lvm_j and the remaining CPU frequency of host h_j. When the lash-up VM lvm_j on host h_j is idle, its CPU resource provision $c(lvm_j)$ will be compressed to a minimum value, and the CPU operating frequency of host h_j can be decreased to save energy, as shown in Fig. 2(a). If the remaining CPU resource rf_j on host h_j is larger than VM lvm_j's CPU resource requirement, the CPU resource provision $c(lvm_j)$ of lash-up VM lvm_j will be immediately scaled up to execute the urgent task, as shown in Fig. 2(b). After the urgent task is finished, the VM lvm_j's CPU frequency and the host's operating frequency will be compressed, as shown in Fig. 2(c).

Step 2. If Step 1 is infeasible and the executing and waiting tasks on some VMs can tolerate the delay of executing the urgent task, transfer these VMs' CPU resource to the lash-up VM for executing the urgent task. After the urgent task is finished, the lash-up VM's CPU resource will be given back. An example in Fig. 3 illustrates this step.

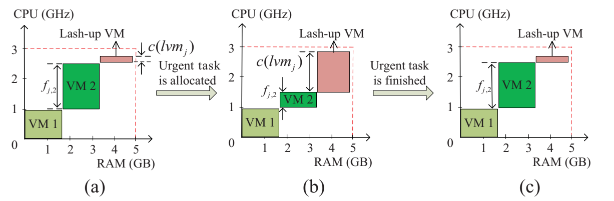

Fig. 3. An example of Step 2

As shown in Fig. 3(a), before allocating the urgent task, the lash-up VM lvm_j, VM1 and VM2 are running on the same physical host, and the lash-up VM occupies negligible CPU resource of this host, while VM1 and VM2 can utilize all the host's CPU resource to run non-urgent tasks that have been scheduled to them. When an urgent task arrives, we here assume that the execution time of the urgent task is very short and all the tasks on VM2 can tolerate the delay caused by executing this urgent task. Step 2 will shift the CPU resource of VM2 to the lash-up VM lvm_j for the urgent task, as shown in Fig. 3(b). After the urgent task is finished, the lash-up VM will return all the CPU resource to VM2, as shown in Fig. 3(c).

Step 3. If the aforementioned two steps are infeasible, select some VMs that all the executing and waiting tasks on them can tolerate the delay of migrating them to other host, then transfer these VMs' CPU resource to the lash-up VM, and migrate these VMs to other host after starting a host.

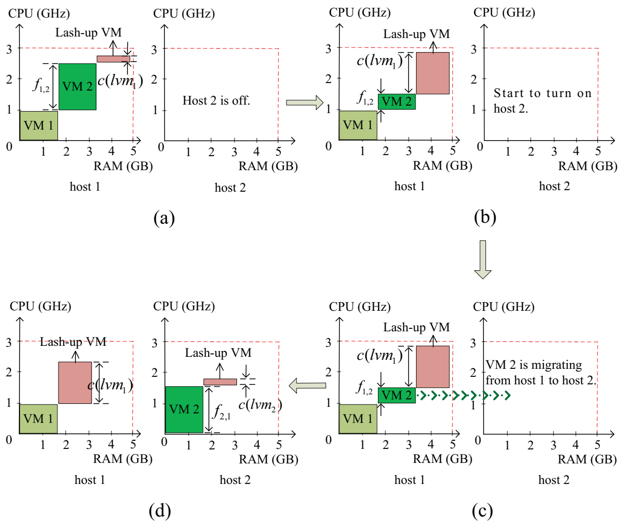

Fig. 4. An example of Step 3

The procedure in Fig. 4 is an example of Step 3. As Fig. 4(a) shows, the lash-up VM lvm_1 with negligible CPU resource collocates with VM1 and VM2 on the host1 before an urgent task arrives. We here assume that the executing and waiting tasks on VM2 cannot tolerate the delay caused by executing the urgent task, but can tolerate the delay of starting host2 and migrating VM2 to host2. As shown in Fig. 4(b), after the urgent task is allocated to VM lvm_1, the CPU resource of VM2 is transferred to the lash-up VM lvm_1, and an closed host (i.e., host2) will be turned on at the same time. When host2 is turned on, it comes to Fig. 4(c), VM2 will be migrated from host1 to host2. After VM2 is migrated to host2, as shown in Fig. 4(d), the CPU resource of VM2 will be increased to satisfy the timing requirements of the executing and waiting tasks on it. Then a new lash-up VM lvm_2 will be created on host2 for the subsequent urgent tasks.

B. Energy efficiencies of hosts

Due to the hosts' heterogeneity, to finish the same workload, different operating frequencies of hosts or using the different hosts will generate different energy consumptions. In order to reduce the energy consumption for cloud data centers, we firstly derive the optimal operating frequencies for heterogenous hosts, and then define the energy efficiencies of these hosts.

Similar to [15], we derive the optimal operating frequencies as follows. Suppose host h_j needs to execute workload Q without deadline. The relationship among execution time (et_j), host operating frequency (f_j) and workload (Q) can be approximated as $et_j = Q/f_j$ [4]. Then, the total energy consumption (tec_j) of host h_j for executing workload Q can be translated as follows:

$$\begin{aligned} tec_j &= \int_{st}^{st+et_j} (s_j \cdot p_j^{max} \cdot c_j^t + \frac{(1-s_j)\cdot p_j^{max}}{(f_j^{max})^3} \cdot (f_j)^3) dt \\ &= s_j \cdot p_{max} \cdot \frac{Q}{(f_j)} + \frac{(1-s_j)\cdot p_{max}}{(f_j^{max})^3} \cdot (f_j)^2 \cdot Q. \end{aligned} \tag{11}$$

The condition of host h_j's optimal frequency is

$$\begin{aligned} &\frac{\partial(tec_j)}{\partial(f_j)} = 0, \text{ i.e.,} \\ &-s_j \cdot p_j^{max} \cdot \frac{Q}{(f_j)^2} + 2 \cdot \frac{(1-s_j)\cdot p_j^{max}}{(f_j^{max})^3} \cdot f_j \cdot Q = 0. \end{aligned} \tag{12}$$

Thus, we have:

$$f_j^* = \sqrt[3]{\frac{s_j}{2\cdot(1-s_j)}(f_j^{max})^3}. \tag{13}$$

Furthermore, since the CPU frequency of host can only operate in a discrete set, i.e., $f_j \in f_j^d = \{f_j^1, f_j^2, \cdots, f_j^{max}\}$, we define the optimal frequency f_j^{opt} of host h_j as below.

Definition 1. The optimal frequency (f_j^{opt}) of host h_j is defined as: (a) $f_j^{opt} = f_j^1$, if $f_j^* \le f_j^1$; (b) $f_j^{opt} = \lceil f_j^* \rceil$, if $f_j^1 \le f_j^* \le f_j^{max}$, where $\lceil f_j^* \rceil$ is the smallest discrete frequency not less than f_j^*; (c) $f_j^{opt} = f_j^{max}$, if $f_j^* \ge f_j^{max}$, i.e.,

$$f_j^{opt} = \begin{cases} f_j^1 & \text{if } f_j^* \le f_j^1, \\ \lceil f_j^* \rceil & \text{if } f_j^1 \le f_j^* \le f_j^{max}, \\ f_j^{max} & \text{if } f_j^* \ge f_j^{max}. \end{cases} \tag{14}$$

Since different hosts with optimal operating frequencies will consume very different energy to finish the same workload. Thus, when scaling up and down the hosts, selecting an effective host is an effective way to reduce energy consumption of the cloud platforms, and we define the energy efficiencies of hosts as follows.

Definition 2. Energy-efficiency EE_j of host h_j is defined as the ratio of the host's optimal frequency (f_j^{opt}) to its energy power when the host operates with the optimal frequency, which can be written as:

$$EE_j = \frac{f_j^{opt}}{s_j \cdot p_j^{max} + \frac{(1-s_j)\cdot p_j^{max}}{(f_j^{max})^3} \cdot (f_j^{opt})^3}. \tag{15}$$

From Definition 2, a host with larger EE_j means the system is more energy efficient, and the hosts with larger EE_j should be preferentially selected to execute the real-time tasks when the system's workload decreases. Oppositely, when the system's workload decreases, the hosts with less EE_j should be turned off first.

C. Scheduling algorithm

Similar to [5], the laxity time of tasks is used to determine tasks' urgency, which is defined as:

Definition 3. The laxity time l_i of task t_i is

$$l_i = d_i - q_i/min\{f_{j,k}\} - ct, \tag{16}$$

where $min\{f_{j,k}\}$ is the VMs' minimal CPU capacity and ct is the current time.

Based on the scheduling architecture in Fig. 1 and the startup-time-aware strategy, we develop the algorithm *STARS* as shown in Algorithm 1.

Algorithm 1: *STARS*: Startup-Times-Aware Scheduling

```
RH ← ∅;
while each new task $t_i$ arrives do
    Delete the mapping of tasks in $RH$ to VMs, and update VMs' ready time;
    Add task $t_i$ to $RH$;
    Sort all the tasks in $RH$ by their laxity time in a non-descending order;
    foreach $t_w \in RH$ do
        $selVM \leftarrow$ select a VM that can finish task $t_w$ before its deadline with the minimum finish time;
        if $selVM$ ==NULL then
            $selVM \leftarrow AddNewVM()$;
        if $selVM$ ==NULL then
            $selVM \leftarrow ScaleUpLashUpVM()$;
        if $selVM$ != NULL then
            Map task $t_w$ to selected VM $selVM$;
        else
            Reject task $t_w$;
```

In order to save energy for virtualized data centers, when deploying new VMs, function $AddNewVM()$ given in Algorithm 2 tries to maintain the operating frequency of each active host as close to its optimal value as possible.

The function $ScaleUpLashUpVM()$, as shown in Algorithm 3, is used to quickly scale up a lash-up VM for an

Algorithm 2: Function AddNewVM()

```
1  Select a kind of VM $vm_k$ with the minimum CPU frequency
   requirement that can finish task $t_w$ before its deadline;
2  $selHost \leftarrow NULL$;   $minFit \leftarrow +\infty$;
3  foreach active host $h_j$ in the system do
4      if $rf_j \geq c(lvm_j)$ & $(pf_j - f_j^{opt})^2 < minFit$ then
5          $selHost \leftarrow h_j$;   $minFit \leftarrow (pf_j - f_j^{opt})^2$;
6  if $selHost$ == $NULL$ then
7      $selHost \leftarrow$ select an off host with the largest $EE_j$ that
       can contain $vm_k$;
8      if $selHost$ != $NULL$ then
9          Turn on the host $selHost$;
10 if $selHost$ != $NULL$ then
11     Create VM $vm_k$ on the selected host $selHost$, denoted as
       $vm_{j,k}$;
12     Return VM $vm_{j,k}$;
13 else
14     Return $NULL$;
```

Algorithm 3: Function ScaleUpLashUpVM()

```
1  Calculate the CPU frequency requirement $c(lvm_j)$ of lash-up
   VM by Eq. (7);
2  foreach active host $h_j$ with idle lash-up VM $lvm_j$ in the
   system do
3      if $rf_j \geq c(lvm_j)$ then
4          Increase the CPU frequency for lash-up VM $lvm_j$;
5          Return lash-up VM $lvm_j$;
6  foreach active host $h_j$ with idle lash-up VM $lvm_j$ in the
   system do
7      if some VMs on host $h_j$ can tolerate the delay of finishing
       task $t_w$ then
8          Transfer these VMs' frequency to lash-up VM $lvm_j$;
9          Return lash-up VM $lvm_j$;
10 foreach active host $h_j$ with idle lash-up VM $lvm_j$ in the
   system do
11     if some VMs on host $h_j$ can tolerate the delay of migration
       then
12         Transfer these VMs' frequency to lash-up VM $lvm_j$;
13         Start an off host and then migrate these non-lash-up
           VMs;
14         Return lash-up VM $lvm_j$;
```

urgent task, and the function $ConsolidateVM()$, as shown in Algorithm 4, will be called to dynamically consolidate VMs and turn off idle physical hosts.

V. Performance Evaluation

To demonstrate the performance improvements gained by *STARS*, we quantitatively compare it with two existing algorithms - *EARH* [4] and *Lowest-DVFS* [7].

The metrics used to evaluate the system performance are guarantee ratio (GR) as (7), total energy consumption (TEC) as (9), resource utilization (RU) as (10) and the change of active hosts' count over time.

Algorithm 4: Function ConsolidateVM()

```
1  $activeHosts \leftarrow$ all the active hosts in the cloud platform;
2  Sort $activeHosts$ by hosts' $EE_j$ in a non-descending order;
3  foreach each $h_j \in activeHosts$ do
4      $candidateHosts \leftarrow activeHosts - h_j$;   $migPlan \leftarrow \phi$;
5      foreach each VM $vm_{j,k}$ on host $h_j$ do
6          $selHost \leftarrow NULL$;   $minValue \leftarrow +\infty$;
7          foreach each host $h_c \in candidateHosts$ do
8              if host $h_c$ can contain VM
               $vm_{j,k}$ & $(pf_j - f_j^{opt})^2 < minValue$ then
9                  $selHost \leftarrow h_c$;
                   $minValue \leftarrow (pf_j - f_j^{opt})^2$;
10         if $selHost$ != $NULL$ then
11             $migPlan \leftarrow migPlan \bigcup (vm_{j,k}, selHost)$;
12         else
13             $migPlan \leftarrow \phi$;   break;
14     if $migPlan$ != $\phi$ then
15         Migrate all the VMs on host $h_j$ according to
           $migPlan$;
16         Shut down host $h_j$, and remove it from $activeHosts$;
```

A. Experimental setup

We compare the three algorithms in the context of Google cloud traces [22]. Based on the analysis in [23], a representative task set, including 955,626 tasks starting from $timestamp$=1,468,890 to $timestamp$=1,559,030, is selected as the testing sample. Fig. 5 (a) and (b) depict the distribution of task count over the time, and the Cumulative Distribution Function (CDF) of the execution times for tasks. From Fig. 5 (a), we can observe that the task count varies remarkably over the time, which means that scaling up and down computing resources according to system's workload is necessary to achieve energy saving. In addition, Fig. 5 (b) shows that the execution time of more than 10% tasks is less than 10s. So, the startup-time-aware polices are necessary to guarantee the deadlines of these tasks with very short execution time.

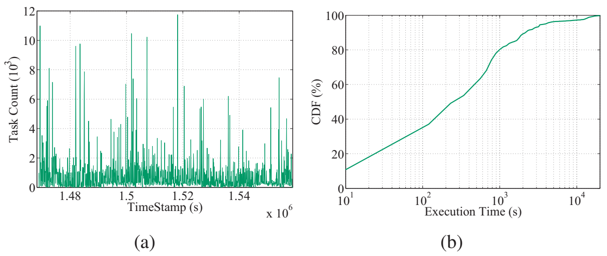

Fig. 5. The characteristics of Google traces.

Since the traces do not contain the information about computation length and deadlines of tasks, we set these two parameters of tasks similar to [23]. The computation length q_i of task t_i is calculated based on the execution duration and the average CPU utilization.

$$q_i = (ts_f - ts_s) \times U_{avg} \times C_{cpu}, \quad (17)$$

where ts_f and ts_s represent the $timestamp$ of finish and schedule event for task t_i; U_{avg} denotes the average CPU

TABLE I
HOST CONFIGURATION PARAMETERS

Host Types	Mem. (GB)	Max Fre. (GHz)	Idle Power (W)	Max Power (W)
Power. R630	64	2.3	51.2	287
RH2288H V2	48	2.4	68.7	137
Altos R380	24	2.2	71.5	316
GT350	12	3.0	79.5	264
Acer F1	16	2.4	88.1	197
Pro. DL160	16	2.5	148	233

usage of this task. The C_{cpu} represents the CPU processing capacity, and we assume C_{cpu}=3000MHz.

We use $deadlineBase$ to control a task's deadline:

$$d_i = a_i + (q_i/2000) \times deadlineBase, \tag{18}$$

where a_i represents the arrival time of t_i.

To simulate the heterogeneous nature of hosts, we choose the configuration parameters of six types of real hosts from 2008 to 2015[1], and set the number of each host type to 600. The relevant parameters of these hosts are shown in Table I. The parameters Max Fre., Idle Power, and Max Power in Table I are respectively corresponding to the variables f_j^{max}, $s_j \cdot p_j^{max}$, and p_j^{max} in (9) and (11). We assume the cloud platform offers four types of VMs, and their CPU resource requirements (in MHz) range from 500 to 2000 with an increment of 500. The startup time of a host is 60s and the creation time of a VM is 30s.

B. Performance impact of task deadlines

Fig. 6 shows the impacts of deadlines on the performance of the three algorithms. We vary the $deadlineBase$ from 1.1 to 3.6 with an increment of 0.5, and select the first 550,000 tasks in the test set for this experiment.

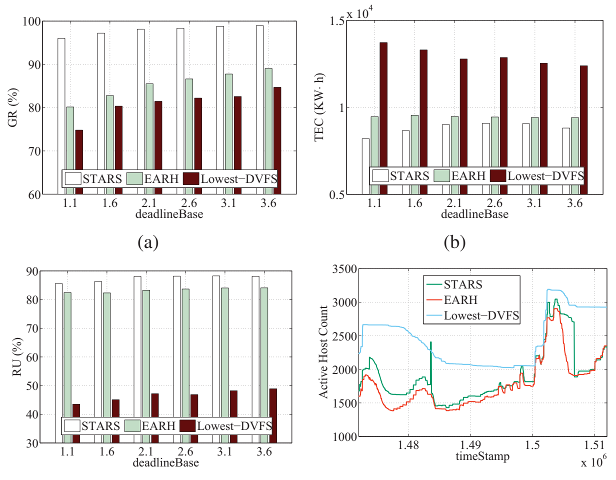

Fig. 6. Performance impact of task deadlines.

We observe from Fig. 6(a) that the guarantee ratios (GR) of the three algorithms increase correspondingly with the increase of $deadlineBase$ (i.e., task deadline becomes looser). This can be explained that as the deadlines of tasks are prolonged, the time overhead of scaling up computing resources has weaker impact on tasks' deadlines. In addition, we can see that *STARS* outperforms *EARH* and *Lowest-DVFS* on average by 13.48% and 19.10% in terms of the guarantee ratio.

Fig. 6(b) shows that, with the increases of $deadlineBase$, the total energy consumptions (TEC) of *STARS* and *EARH* keep stable around at 8805 KW·h and 9461 KW·h, while that of *Lowest-DVFS* decreases significantly from 13727 to 12389 KW·h. With respect to energy consumption, *STARS* outperforms *EARH* on all the instances. This can be attributed to the following two reasons. Firstly, *STARS* gives higher priority to use the hosts with higher efficiencies when scaling up and down hosts. Then, when deploying VMs to hosts, *STARS* strives to make the operating frequency of each active host as close to its optimal frequency as possible.

Fig. 6(c) shows that the resource utilization (RU) of the three algorithms increase correspondingly with the increases of $deadlineBase$, and this trend is especially outstanding with *Lowest-DVFS*. As tasks' deadlines become looser, more VMs with less CPU requirement will be utilized, which is helpful for increasing the resource utilization of active hosts. With respect to resource utilization, *STARS* on average outperforms *EARH* and *Lowest-DVFS* by 4.71% and 46.69%, respectively.

Fig. 6(d) depicts the fluctuation of active host count over time for *STARS*, *EARH* and *Lowest-DVFS*. We can observe from Fig. 6(d) that the active host counts of *STARS* and *EARH* are dynamically varied according to the system's workload, while that of *Lowest-DVFS* experiences big rise and fall because both *STARS* and *EARH* utilize the resources consolidation strategies to reduce active hosts when the system's workload decreases, but *Lowest-DVFS* has no the ability.

C. Performance impact of task count

We conduct a group of experiments to analyze the impact of task count on the performance of the three algorithms. Fig. 7 illustrates the experimental results when the count of tasks varies from 450,000 to 950,000 with an increment of 100,000, and the $deadlineBase$ is set to be 2.1.

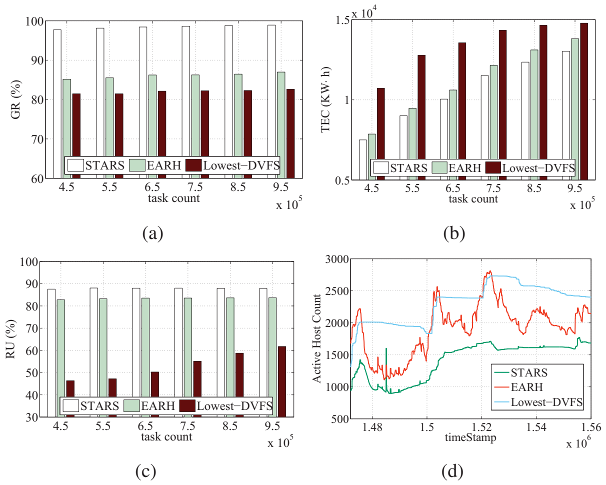

Fig. 7. Performance impact of task count.

In Fig. 7(a), we can see that the GRs for *STARS*, *EARH* and *Lowest-DVFS* are respectively stable at 98.43%, 86.10% and

[1]http://www.spec.org/power_ssj2008/results/

82.03%. Since there are infinite resources in clouds, when task count increases, more hosts will be started up to execute more tasks. However, not all the tasks' deadlines can be guaranteed although there are enough resources because starting a new host or creating a new VM needs additional time overhead, which may violate some urgent tasks' timing constraints. In addition, *STARS* improves the GRs of *EARH* and *Lowest-DVFS* by an average of 12.53% and 16.66%, respectively.

Fig. 7(b) reveals that the TEC of the three tested algorithms linearly increases with the count of tasks. This is because the guarantee ratios of these three algorithms vary slightly around different constants; the total tasks' computation lengths are linear to the number of tasks and the TEC of the system is almost linear to the total computation length. On average, *STARS* consumes less energy than *EARH* and *Lowest-DVFS* by 5.56% and 29.40%, respectively.

The experimental results in Fig. 7(c) are consistent with that in Fig. 6(c), so does the explanation.

The fluctuation trend of the three algorithms in Fig. 7(d) is similar to that in Fig. 6(d), so does the explanation. From Fig. 7(d), we can also find that the active host count of the three algorithms is much lower than that in Fig. 6(d). The reason is that the deadlines of real-time tasks become longer, which allows less hosts to work longer to finish the total workload while guaranteeing tasks' timing requirements.

VI. Conclusions

In this paper, we focus on the time overheads of scaling up resources delay the start and even violate the deadlines of real-time tasks. We have presented a startup-time-aware scheduling architecture and the *EARH* scheduling algorithm that can make good trade-offs between the guaranteeing ratio of real-time tasks and the energy consumption of the overall system. We evaluate the effectiveness of *STARS* through simulation experiments on Google workload traces, showing the effectiveness of the *STARS* approach, compared with two existing known energy-saving scheduling algorithms.

Acknowledgements

This research was supported by the National Natural Science Foundation of China under grants No. 61572511 and No. 11428101, the Scientific Research Project of National University of Defense Technology under grant ZK16-03-57 and ZK16-03-09 as well as the Southwest Electron & Telecom Technology Institute under Grant No. 2015014. The author from Georgia Tech is partially support by the National Science Foundation under grants CNS-1115375, NSF 1547102, SaTC 1564097 and Intel ISTC on Cloud Computing.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, pp. 50–58, 2010.

[2] J. Koomey, "Growth in data center electricity use 2005 to 2010," *A report by Analytical Press, completed at the request of The New York Times*, p. 9, 2011.

[3] C. Pettey, "Gartner estimates ict industry accounts for 2 percent of global co2 emissions," *Dostupno na: https://www. gartner. com/newsroom/id/503867*, vol. 14, p. 2013, 2007.

[4] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 168–180, 2014.

[5] H. Chen, X. Zhu, H. Guo, J. Zhu, X. Qin, and J. Wu, "Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment," *Journal of Systems and Software*, vol. 99, pp. 20–35, 2015.

[6] 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, no. 5, pp. 755–768, 2012.

[7] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of cloud resources for real-time services," in *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*. ACM, 2009, pp. 1–6.

[8] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya *et al.*, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in computers*, vol. 82, no. 2, pp. 47–111, 2011.

[9] D. Hagimont, C. M. Kamga, L. Broto, A. Tchana, and N. De Palma, "Dvfs aware cpu credit enforcement in a virtualized system," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2013, pp. 123–142.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 164–177.

[11] L. Cherkasova, D. Gupta, and A. Vahdat, "When virtual is harder than real: Resource allocation challenges in virtual machine based it environments," *Hewlett Packard Laboratories, Tech. Rep. HPL-2007-25*, 2007.

[12] Q. Liang, J. Zhang, Y.-H. Zhang, and J.-M. Liang, "The placement method of resources and applications based on request prediction in cloud data center," *Information Sciences*, vol. 279, pp. 735–745, 2014.

[13] J. Wu, "Energy-efficient scheduling of real-time tasks with shared resources," *Future Generation Computer Systems*, vol. 56, pp. 179–191, 2016.

[14] C.-H. Hsu, K. D. Slagter, S.-C. Chen, and Y.-C. Chung, "Optimizing energy consumption with task consolidation in clouds," *Information Sciences*, vol. 258, pp. 452–462, 2014.

[15] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 732–749, 2011.

[16] Y. C. Ren, J. Suzuki, S. Omura, and R. Hosoya, "Leveraging active-guided evolutionary games for adaptive and stable deployment of dvfs-aware cloud applications," *International Journal of Software Engineering and Knowledge Engineering*, vol. 25, no. 05, pp. 851–870, 2015.

[17] Y. Zhang, Y. Wang, and C. Hu, "Cloudfreq: Elastic energy-efficient bag-of-tasks scheduling in dvfs-enabled clouds," in *Proceedings of the 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2015, pp. 585–592.

[18] C. M. Kamga, G. S. Tran, and L. Broto, "Extended scheduler for efficient frequency scaling in virtualized systems," *ACM SIGOPS Operating Systems Review*, vol. 46, no. 2, pp. 28–35, 2012.

[19] Y. Ding, X. Qin, L. Liu, and T. Wang, "Energy efficient scheduling of virtual machines in cloud with deadline constraint," *Future Generation Computer Systems*, vol. 50, pp. 62–74, 2015.

[20] D. G. d. Lago, E. R. Madeira, and L. F. Bittencourt, "Power-aware virtual machine scheduling on clouds using active cooling control and dvfs," in *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*. ACM, 2011, pp. 1–6.

[21] V. Hanumaiah and S. Vrudhula, "Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 349–360, 2014.

[22] https://code.google.com/p/googleclusterdata/wiki, "Google cluster data v2."

[23] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "An approach for characterizing workloads in google cloud to derive realistic resource utilization models," in *Proceedings of the 7th International Symposium on Service Oriented System Engineering (SOSE)*. IEEE, 2013, pp. 49–60.