The Journal of Systems and Software 99 (2015) 20–35

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment

Huangke Chen [a], Xiaomin Zhu [a,*], Hui Guo [b], Jianghan Zhu [a], Xiao Qin [c], Jianhong Wu [d]

[a] Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha 410073, PR China
[b] School of Computer Science and Engineering, University of New South Wales, NSW 2052, Australia
[c] Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849-5347, USA
[d] Department of Mathematics and Statistics, York University, Toronto M3J1P3, Canada

ARTICLE INFO

Article history:
Received 13 January 2014
Received in revised form 4 July 2014
Accepted 28 August 2014
Available online 6 September 2014

Keywords:
Green cloud computing
Uncertain scheduling
Proactive and reactive

ABSTRACT

Green cloud computing has become a major concern in both industry and academia, and efficient scheduling approaches show promising ways to reduce the energy consumption of cloud computing platforms while guaranteeing QoS requirements of tasks. Existing scheduling approaches are inadequate for real-time tasks running in uncertain cloud environments, because those approaches assume that cloud computing environments are deterministic and pre-computed schedule decisions will be statically followed during schedule execution. In this paper, we address this issue. We introduce an interval number theory to describe the uncertainty of the computing environment and a scheduling architecture to mitigate the impact of uncertainty on the task scheduling quality for a cloud data center. Based on this architecture, we present a novel scheduling algorithm (PRS[1]) that dynamically exploits proactive and reactive scheduling methods, for scheduling real-time, aperiodic, independent tasks. To improve energy efficiency, we propose three strategies to scale up and down the system's computing resources according to workload to improve resource utilization and to reduce energy consumption for the cloud data center. We conduct extensive experiments to compare PRS with four typical baseline scheduling algorithms. The experimental results show that PRS performs better than those algorithms, and can effectively improve the performance of a cloud data center.

© 2014 Elsevier Inc. All rights reserved.

1. Introduction

Cloud computing has become a paradigm for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell and Grance, 2009). To satisfy such a soaring demand of computing services, IT companies (e.g., Google and Facebook) are rapidly deploying distributed data centers in different administrative domains around the world.

Consequently, tens of thousands of hosts in these data centers consume enormous energy for computing and equipment cooling operations (Luo et al., 2012). It is reported that the energy consumed in data centers is about 1.5% of the global electricity in 2010, and the percentage will be doubled by 2020 if the current trends continue (Koomey, 2011). Apart from the operating cost, high energy consumption will result in low reliability of the system since the failure rate of hosts doubles for every 10-degree increase in temperature (Cameron et al., 2005). In addition, high energy consumption has a negative impact on environment because generating electrical energy from fossil fuels produces a large amount of CO_2 emissions, which are estimated to be 2% of the global emissions (Pettey, 2007). Therefore, reducing energy consumption or conducting green computing has become a grand challenge when deploying and operating cloud data centers.

With the development of virtualization technology (Barham et al., 2003), a single physical host can run multiple virtual machines (VMs) simultaneously. In addition, the VMs can be relocated by live operations, such as VM creation, VM live migration and VM deletion, to achieve fine-grained optimization of computing resources for cloud data centers. This technology offers significant opportunities for green computing (Beloglazov et al., 2012). Leveraging

* Corresponding author.
E-mail addresses: hkchen@nudt.edu.cn (H. Chen), xmzhu@nudt.edu.cn (X. Zhu), huig@cse.unsw.edu.au (H. Guo), jhzhu72@gmail.com (J. Zhu), xqin@auburn.edu (X. Qin), wujh@mathstat.yorku.ca (J. Wu).

[1] Proactive and Reactive Scheduling.

http://dx.doi.org/10.1016/j.jss.2014.08.065
0164-1212/© 2014 Elsevier Inc. All rights reserved.

the capabilities of virtualization technology, one can scale up or down VMs rapidly according to the current workloads in the system. When the system is overloaded, more VMs are added; when the system is underloaded, the VMs can be consolidated to a minimal number of physical hosts and the idle hosts can be turned off. Hosts in a completely idle state can dissipate over 70% as much power as when they fully utilized (Ma et al., 2012). Turning-off idle hosts, therefore, means significant power savings.

Nevertheless, the virtualization also brings about new challenges to the resource management in clouds due to the fact that multiple VMs can share the hardware resources (e.g., CPU, memory, I/O, network, etc.) of a physical host (Kong et al., 2011). The resource sharing may cause the performance of VMs subjecting to considerable uncertainties in cloud computing environments mainly due to I/O interference between VMs (Bruneo, 2014; Armbrust et al., 2010) and hosts are overloaded (Beloglazov and Buyya, 2013). For example, the ready time and the computing capacity of a VM arbitrarily varies over time, which makes it difficult to accurately measure the execution timing parameters and resource usage of VMs. Such dynamic and non-deterministic characteristics of the VM computing cause great difficulties for efficient resource management in clouds.

In addition, a primary fraction of computing applications in cloud data centers are real-time tasks. The arrival times of these tasks are dynamic and the predictions of their execution duration can also be difficult and sometimes impossible (Van den Bossche et al., 2010), since most real-time tasks are fresh and no much information is available to help the accuracy of the predictions. The imprecise execution prediction and dynamic task arrival time leave the associated scheduling timing constraints (i.e., start time, execution time and finish time) under considerable uncertainty. Furthermore, real-time tasks often need deadlines to guarantee their timing requirements, which further exacerbates the problem of efficient task scheduling and resource management.

Motivation: Due to the dynamic and uncertain nature of cloud computing environments, numerous schedule disruptions (e.g., shorter or longer than expected task execution time, variation of VM performance, arrival of urgent tasks, etc.) may occur and the pre-computed baseline schedule may not be executed and may not be effective in real execution. Unfortunately, the vast majority of researches did not consider the uncertainties of clouds, which may leave a large gap between the real execution behavior and the behavior initially expected. To address this issue, we study how to describe these uncertain parameters, how to control uncertainties' impact on scheduling results, and how to reduce the energy consumption in cloud data centers, while guaranteeing the timing requirements of real-time tasks.

Contributions: The major contributions of this work are:

- An uncertainty-aware architecture for scheduling real-time tasks in the cloud computing environment.
- A novel algorithm named PRS that combines proactive with reactive scheduling methods for scheduling real-time tasks and computing resources when considering the uncertainties of the system.
- Three system scaling strategies according to dynamic workloads to reduce energy consumption.
- The experimental verification of the proposed PRS algorithm based on randomly generated test instances and real world traces from Google.

The remainder of this paper is organized as follows. The related work in the literature is summarized in Section 2. Section 3 presents the scheduling model and the problem formulation. The energy-aware scheduling algorithm for real-time tasks considering the uncertainties of the system is introduced in Section 4. Section 5 conducts extensive experiments to evaluate the performance of our algorithm by comparing it with four baseline scheduling algorithms. Section 6 concludes the paper with a summary and future directions.

2. 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 number of energy-aware scheduling algorithms have been developed. Among them, there are two typical approaches. One is DVFS (dynamic voltage and frequency scaling) based and another is machine virtualization based.

The DVFS technique makes trade-offs between processor power and performance and has been commonly used to reduce the power consumption of data centers. For example, Garg et al. (2011) proposed near-optimal energy-efficient scheduling policies that leverages DVFS to scale down the CPU frequency as far as possible to minimize the carbon emission and maximize the profit of the cloud providers. Li and Wu (2012) proposed a Relaxation-based Iterative Rounding Algorithm (RIRA) for DVFS-enabled heterogeneous multiprocessor platforms to minimize overall energy consumption while meeting tasks deadlines. Rizvandi et al. (2011) focused on the issue of high energy consumption in cluster, and presented the MVFS-DVFS algorithm to fully utilize slack times and reduce energy consumption on processors. Zhu et al. (2013) proposed an energy-efficient elastic (3E) scheduling strategy to make trade-offs between users' expected finish time and energy consumption by adaptively adjusting CPU's supply voltages according to the system workload. However, the DVFS is mainly implemented on host processor machines and their energy consumption contributes about one-third of the total system power (Ahmad and Vijaykumar, 2010). In addition, only the dynamic power (about 30% of the processor power Beloglazov et al., 2012) can be moderated by DFVS. Due to those limitations of DVFS, the virtualization technique, used to consolidate VMs for low energy consumption of data centers, is becoming popular and is the focus in this paper.

The vast majority of the energy-aware scheduling research efforts over the past several years have concentrated on dynamical consolidation of VMs according to the system workload, to reduce the number of physical hosts so that the idle hosts can be switched off for low energy consumption. Hermenier et al. examined the overhead of migrating a VM to its chosen destination, and proposed a resource manager named Entropy for homogeneous clusters. The Entropy can dynamically consolidate VMs based on constraint programming and explicitly takes into account the cost of the migration plan (Hermenier et al., 2009). Younge et al. (2010) developed a novel green framework where green computing was realized by energy efficient scheduling and VM management components. Srikantaiah et al. (2008) explored the inter-relationships among energy consumption, resource utilization, and performance of consolidated workloads, and designed a heuristic scheme for minimizing system energy consumption while meeting the performance constraint. Hsu et al. (2014) studied how to dynamically consolidate tasks to increase resource utilization and reduce energy consumption, and presented an energy-aware task consolidation (ETC) method to optimize energy usage in cloud systems. Our work also leverages the VM consolidation technique to reduce the systems' energy consumption. However, unlike the above existing approaches, where uncertainties of tasks' execution times and VMs' performance were not considered, our design takes the uncertainties into account, and we employ proactive and reactive methods to mitigate the impact of uncertainties on the scheduling quality for cloud data centers.

There also exist some work investigating the task scheduling strategies under uncertain computing environments. Qiu et al. studied the problem of assigning computing units to each task in a system to achieve energy savings at a minimum cost. The system is modeled by a Probabilistic Data Flow Graph (PDFG) and the timing constraint of the system was satisfied with a guaranteed confidence probability (Qiu and Sha, 2009). Li et al. (2011) studied the impacts of inaccurate execution time information on the performance of resource allocation, and presented an evaluation method to compare the performance of three widely used greedy heuristics. Xian et al. (2008) presented an approach to combine intra- and intertask voltage scheduling for energy reduction in hard real-time systems assuming that the probabilistic distributions of tasks' execution time are available. Kong et al. (2011) focused on uncertain availabilities of virtualized server nodes and workloads, and utilized the type-I and type-II fuzzy logic systems to predict the availability of resources and workloads to improve performance of virtualized data centers. The algorithms in these work need the probability distributions or membership functions of tasks' execution times for deterministic proactive schedule decisions. Unlike the aforementioned approaches, the algorithm in this paper firstly builds proactive baseline schedules; the proactive baseline schedules are then dynamically repaired when disruptions (e.g., the urgent tasks arrive, systems' load become too heavy, and the like) occur during the course of executions.

3. Modeling and formulation

In this section, we firstly introduce the theory of interval number for description of uncertain parameters encountered in the task scheduling, then propose a scheduling architecture for cloud data centers. Based on this architecture, we form our scheduling problem.

3.1. Interval number and arithmetic operations

Interval number can be used to specify imprecise, uncertain and incomplete data and/or decision variables (Sengupta and Pal, 2009). The related definitions and operations that will be used in our task scheduling are given below.

Definition.1. (Sengupta and Pal, 2009) Let R be the set of all real numbers, and $a^-, a^+ \in R$. An interval number is defined as follows: $\widetilde{a} = [a^-, a^+] = \{t | a^- \le t \le a^+, t \in R\}$, where a^- and a^+ are the lower and upper bounds of the interval number $\widetilde{a}$, respectively. If $a^- = a^+ = t$, then $\widetilde{a} = [t, t] = t$ is a real number. We denote the set of all interval numbers by $I(R)$. The interval between the lower and upper bounds of an uncertain number reflects its uncertainty degree.

Definition.2. (Sengupta and Pal, 2009) If $\widetilde{a} = [a^-, a^+], \widetilde{b} = [b^-, b^+] \in I(R)$, then the addition operation $\oplus$ and the subtraction operation $\ominus$ between the two interval numbers are defined as follows.

$$\begin{aligned} &\widetilde{a} \oplus \widetilde{b} = [a^- + b^-, a^+ + b^+]; \\ &\widetilde{a} \ominus \widetilde{b} = [a^- - b^+, a^+ - b^-]. \end{aligned} \tag{1}$$

Inference 1. Given a set of interval numbers $\widetilde{a}_1 = [a_1^-, a_1^+], \widetilde{a}_2 = [a_2^-, a_2^+], \ldots, \widetilde{a}_n = [a_n^-, a_n^+]$, then the sum of these interval numbers is:

$$\sum_{i=1}^{n} \widetilde{a}_i = \sum_{i=1}^{n} [a_i^-, a_i^+] = \left[\sum_{i=1}^{n} a_i^-, \sum_{i=1}^{n} a_i^+ \right]. \tag{2}$$

By Inference 1, we know that the interval of $\sum_{i=1}^{n} \widetilde{a}_i$ is the sum of the interval of each interval number. The uncertainty increases with the count of interval numbers.

Definition.3. (Sengupta and Pal, 2009) If $\widetilde{a} = [a^-, a^+], \widetilde{b} = [b^-, b^+] \in I(R)$, then the multiplication operation $\otimes$ and the division operation ø between the two interval numbers are defined as follows.

$$\begin{aligned} &\widetilde{a} \otimes \widetilde{b} = [\min\{a^- b^-, a^- b^+, a^+ b^-, a^+ b^+\}, \\ &\quad \max\{a^- b^-, a^- b^+, a^+ b^-, a^+ b^+\}]; \\ &\widetilde{a} ø \widetilde{b} = [a^-, a^+] \otimes [1/b^-, 1/b^+]. \end{aligned} \tag{3}$$

We use interval numbers to model the uncertain execution timing parameters in the cloud computing. Our optimization problem of task scheduling can therefore be formed based on those parameters, which will be detailed in the next section.

3.2. System architecture and scheduling model

In this paper, the targeted system is a large-scale data center consisting of n heterogeneous physical hosts $H = \{h_1, h_2, \ldots, h_n\}$. Each host is characterized by $h_j = \{c_j, m_j, s_j, p_j, VM_j\}$, where c_j is the host CPU performance measured in Millions Instructions Per Second (MIPS), m_j the memory size, s_j the storage capacity, and p_j the energy consumption when the host is fully utilized; a set of VMs on the host are denoted by VM_j, $VM_j = \{vm_{jk}, k = 0, 1, \ldots, |VM_j|\}$, and vm_{jk} is the kth virtual machine (VM) on host h_j. A VM is, in turn, modeled as $vm_{jk} = \{\widetilde{c}_{jk}, m_{jk}, s_{jk}, wtk_{jk}, etk_{jk}\}$, where $\widetilde{c}_{jk}$, m_{jk} and s_{jk} are respectively the CPU performance (in MIPS), memory and storage capacity required for VM vm_{jk}; the wtk_{jk} and etk_{jk} represent the waiting and executing task on VM vm_{jk}, respectively. Note that the CPU performance of VMs arbitrarily varies over time, and their lower and upper bounds can be gained before scheduling, e.g., by utilizing the Markov Chain prediction method (Beloglazov and Buyya, 2013), so we utilize interval number to present them.

In this paper, we focus on real-time, aperiodic, independent tasks, denoted as $T = \{t_1, t_2, \ldots, t_m\}$. For a certain task $t_i \in T$, it can be modeled by $t_i = \{a_i, d_i, \widetilde{l}_i, dz_i\}$, where a_i, d_i, $\widetilde{l}_i$ and dz_i represent the arrival time, deadline, length (in MI) and data size (in MB) of task t_i, respectively. Note that the length of a task is uncertain before scheduling, and its lower and upper bounds can be gained, e.g., via machine learning method proposed in Berral et al. (2011). So we employ interval number to describe the computing length of a task.

Each real-time task has a level of urgency, which can be determined in many ways, for example, deadline (Mills and Anderson, 2010), and laxity (Oh and Yang, 1998). Here we use the laxity for the task urgency, which is given below.

Definition.4. The laxity (Oh and Yang, 1998) L_i of task t_i is

$$L_i = d_i - \frac{l_i^+}{\min\{c_{jk}^-\}} - ct. \tag{4}$$

where l_i^+ and d_i are the computation length upper bound and deadline of task t_i, respectively; $\min\{c_{jk}^-\}$ is the computing capacity lower bound of the VM with minimal CPU performance; $l_i^+ / \min\{c_{jk}^-\}$ represents the upper bound of task t_i's maximal execution time and ct is the current time.

Comparing with the deadline-only-based urgency used in the traditional Earliest Deadline First (EDF) (Mills and Anderson, 2010) policy the laxity proposed here can better reflect tasks' urgency by considering both their computation lengths and deadlines since the processing requirements of these tasks are heterogeneous. The task with the smallest laxity should be first considered for execution

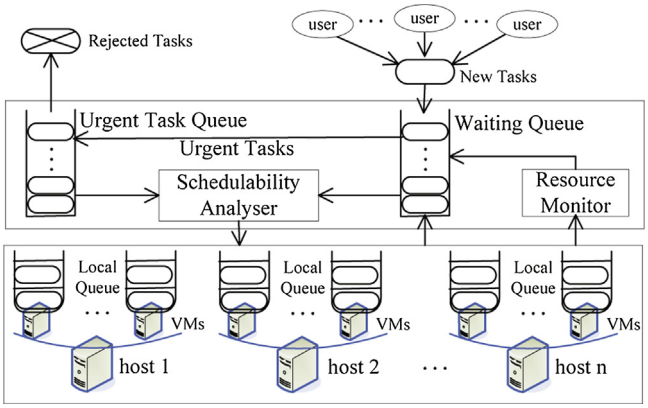


Fig. 1. The uncertainty-aware scheduling architecture.

when scheduling. If L_i is negative, task t_i cannot be successfully completed before its deadline d_i.

Definition.5. Urgent task: a task becomes urgent when its laxity L_i is equal to or less than a preestablished threshold L_d. In this paper, let L_d be the time for turning on a host and creating a VM on it (e.g., $L_d = 120$ s).

The scheduling architecture of a cloud data center introduced in this paper is shown in Fig. 1. Similar to traditional multiprocessor systems (Ma et al., 2012), the scheduling model in cloud data centers consists of three layers: user layer, scheduling layer and resource layer. However, the most difference between these two systems are that the resource layer in cloud data centers can be further divided into two layers: host layer and virtual machine (VM) layer, and that the VM layer can be scaled up and down according to the workload in the cloud data centers.

The scheduler consists of a waiting queue (WQ), an urgent task queue (UTQ), a resource monitor, and a schedulability analyzer. The WQ accommodates both new tasks and waiting tasks to be scheduled, while the UTQ holds all the urgent tasks that should be scheduled immediately. The resource monitor detects the system's status, and the schedulability analyzer is responsible for scheduling task and dynamically allocating computing resources. In addition, any VM vm_{jk} has a local queue (LQ), which holds the executing task etk_{jk} and the waiting task wtk_{jk} that have been scheduled on this VM.

The overview of the scheduling process for this architecture is as follows. When non-urgent task arrives, it will be added into the waiting queue (WQ), and the tasks in WQ are arranged in an ascending order by their laxity. When urgent task arrives or some tasks in WQ become urgent over time, they will be delivered to UTQ directly, and these tasks will be scheduled to VMs as executing or waiting tasks immediately. In addition, when any one of the two following cases occurs, some tasks in WQ will be scheduled to VMs.

- The first case is that when the arrival rate of tasks increases and the resources required for tasks in WQ exceed the initiated computing resources in the system, which can be approximated as Formula (5), computing resources will be scaled up for the new arrival tasks.

$$\exists t_i \text{ in } WQ,\ \sum_{j=1}^{n}\sum_{k=1}^{|VM_j|} c_{jk}^{-}(d_i - \max(rt_{jk}^{+}, ct)) < \sum_{t_s \in T_i} l_s^{+}. \tag{5}$$

  where rt_{jk}^{+} is the ready time's upper bound of VM vm_{jk}, i.e., the time that VM vm_{jk} will finish all the tasks that have been mapped to it, $T_i = \{t_s | d_s \le d_i\}$ represents the subset of tasks in WQ whose deadlines are smaller or equal to the deadline of task t_i, and l_s is the computing length of task t_i. Besides, $\sum_{j=1}^{n}\sum_{k=1}^{|VM_j|} c_{jk}^{-}(d_i - \max(rt_{jk}^{+}, ct))$ represents the available computing resources before task t_i's deadline d_i, and $\sum_{t_s \in T_i} l_s^{+}$ represents the computing resources required by the subset of tasks $T_i = \{t_s | d_s \le d_i\}$.
- The second case is when a VM finishes a task. On a task completion by a VM, the waiting task on the VM starts to execute immediately, and searching a new waiting task for the VM from WQ will be performed.

The unique features in this scheduling architecture are that most of waiting tasks are waiting in the WQ instead of waiting on the LQs of VMs and at most one task is allowed to wait on the LQ of each VM. The benefits of this scheduling architecture are summarized as follows.

- It can prohibit propagation of uncertainties throughout the schedule. According to Inference 1, we know that as the count of waiting tasks increases, the uncertainty degrees of waiting tasks become larger, which will significantly affect the stability of schedule. Therefore, we need to control the count of tasks waiting on LQs (i.e., waiting on VM directly) to prevent the propagation of uncertainties.
- This design allows each task waiting on LQ to start as soon as its preceding task has finished, so the possible execution delay for a new task is removed.
- This design enables overlapping of communications and computations. When VM is executing a task and the LQ is empty, VM can simultaneously receive another task as a waiting task. By doing so, communications and computations are efficiently overlapped to save time, and overlapping of communications with computations has been proved to be an efficient method to improve scheduling performance (Hu and Veeravalli, 2013).
- It also can reduce the overheads of task transfer among hosts when corresponding VMs need to migrate. The reason is that when consolidate VMs by live migrations, the waiting tasks on VMs will also be migrated with VMs, which may cause overheads. But this scheduling architecture maintains most waiting tasks in WQ instead of LQs of VMs, and thus reduces the moving tasks' data among hosts during VM migrations.

3.3. Problem formulations

As host resources are limited, the amount of resources required for VMs on a host must not be greater than the capacity of the host. The requirement forms the first scheduling constraint, as can be formally expressed below.

$$\begin{aligned} c_j \ominus \sum_{k=1}^{|VM_j|} \widetilde{c}_{jk} \ge 0, &\quad \forall h_j \in H; \\ m_j - \sum_{k=1}^{|VM_j|} m_{jk} \ge 0, &\quad \forall h_j \in H; \\ s_j - \sum_{k=1}^{|VM_j|} s_{jk} \ge 0, &\quad \forall h_j \in H. \end{aligned} \tag{6}$$

We utilize assignment variable x_{ijk} to reflect the mapping of task t_i to VM vm_{jk} on host h_j in a cloud data center. The assignment

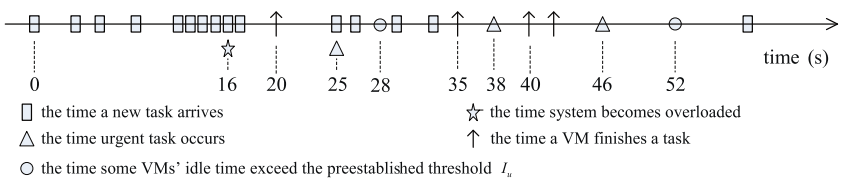


Fig. 2. The reactive disruptions in a time axis.

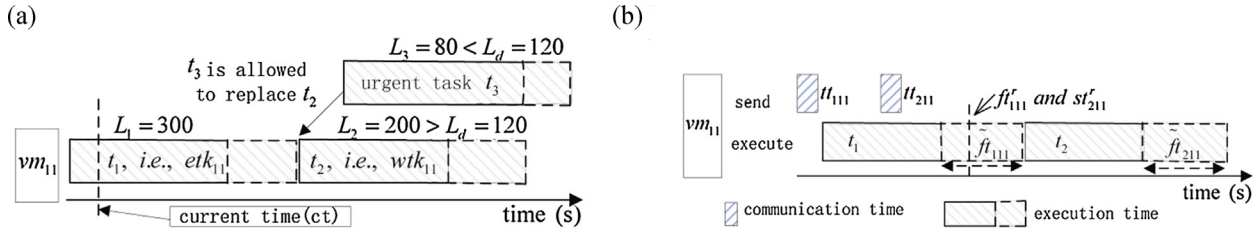


Fig. 3. The illustration of property 3, 4 and 5.

variable x_{ijk} is 1 when task t_i is assigned to VM vm_{jk}, otherwise, x_{ijk} equals 0, i.e.,

$$x_{ijk} = \begin{cases} 1, & \text{if } t_i \text{ is assigned to } vm_{jk}, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

We let $\widetilde{st}_{ijk}$, $\widetilde{et}_{ijk}$ and $\widetilde{ft}_{ijk}$ be the start time, execution time and finish time of task t_i on VM vm_{jk}, respectively. Since the CPU performance $\widetilde{c}_{jk}$ of VM vm_{jk} vary over the time and the total amount of computation length $\widetilde{l}_i$ of task t_i cannot be exactly determined before scheduling, the parameters (i.e., $\widetilde{st}_{ijk}$, $\widetilde{et}_{ijk}$, $\widetilde{ft}_{ijk}$, $\widetilde{c}_{jk}$, and $\widetilde{l}_i$) are uncertain and we utilize the interval number to describe these uncertain parameters (e.g., $\widetilde{l}_i = [l_i^-, l_i^+]$ and $\widetilde{c}_{jk} = [c_{jk}^-, c_{jk}^+]$ etc). For $l_i^-, l_i^+, c_{jk}^-, c_{jk}^+ > 0$, the execution time of task t_i on VM vm_{jk} can be calculated via Definition.3 as follows.

$$\begin{aligned} \widetilde{et}_{ijk} &= \widetilde{l}_i \text{ø} \widetilde{c}_{jk} = [l_i^-, l_i^+] \text{ø} [c_{jk}^-, c_{jk}^+] = [l_i^-, l_i^+] \otimes [1/c_{jk}^-, 1/c_{jk}^+] \\ &= [\min\{l_i^-/c_{jk}^-, l_i^-/c_{jk}^+, l_i^+/c_{jk}^-, l_i^+/c_{jk}^+\}, \max\{l_i^-/c_{jk}^-, l_i^-/c_{jk}^+, \\ &\quad l_i^+/c_{jk}^-, l_i^+/c_{jk}^+\}] = [l_i^-/c_{jk}^+, l_i^+/c_{jk}^-]. \end{aligned} \tag{8}$$

The finish time $\widetilde{ft}_{ijk}$ of task t_i on VM vm_{jk} can be easily determined as follows.

$$\widetilde{ft}_{ijk} = \widetilde{st}_{ijk} \oplus \widetilde{et}_{ijk}. \tag{9}$$

In turn, the finish time $\widetilde{ft}_{et_{jk}}$ of the executing task et_{jk} on the VM vm_{jk} determines the start time $st_{wt_{jk}}$ of the waiting task wt_{jk} on the VM vm_{jk}, which can be described as follows.

$$\widetilde{st}_{wt_{jk}} = \widetilde{ft}_{et_{jk}}. \tag{10}$$

In addition, let ft^r_{ijk} be the actual finish time of task t_i on VM vm_{jk}, which can be any value in the interval, $\widetilde{ft}_{ijk}$. For example, assume the estimated finish time of task t_1 on VM vm_{11} is $\widetilde{ft}_{111} = [100, 150]$ s before scheduling; the actual finish time ft^r_{111} should be between 100 s and 150 s (e.g., 120 s) after the task t_1 has been finished on VM vm_{11}. Under the uncertain cloud computing scheduling environments, it is the actual finish time ft^r_{ijk} that determines whether the task's timing requirement has been guaranteed or not. So, we introduce the status variable o_{ijk} to record whether the timing requirement of task t_i on VM vm_{jk} has been guaranteed or not. If a task t_i is assigned to VM vm_{jk} and its actual execution time is smaller or equal to its deadline, the time requirement of this task is guaranteed, as specified in Formula (11).

$$o_{ijk} = \begin{cases} 1, & \text{if}((ft^r_{ijk} \le d_i) \text{and} (x_{ijk} = 1)), \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

Since only one result of each task is needed, the second constraint:

$$\sum_{j=1}^{n} \sum_{k=1}^{|VM_j|} o_{ijk} \le 1, \quad \forall t_i \in T. \tag{12}$$

Subjecting to aforementioned constraints, the primary optimization objective is to maximize the ratio of tasks finished before their deadlines, which can be represented as follows.

$$\text{Maximize} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{|VM_j|} \frac{o_{ijk}}{m}. \tag{13}$$

where m and n are the count of tasks submitted and hosts in the system, respectively.

Another objective needed to be optimized under uncertain environments is to minimize the stability cost function $\sum w_i(|st^+_{ijk} - st^r_{ijk}|)$ (Van de Vonder et al., 2008), defined as the weighted sum of the absolute deviations between the predicted starting time's upper bound st^+_{ijk} of task t_i in the baseline and their true starting times st^r_{ijk} during actual execution, which can be described as follows.

$$\text{Minimize} \sum_{i=1}^{m} w_i(|st^+_{ijk} - st^r_{ijk}|). \tag{14}$$

where w_i represents the marginal cost of time deviation between the predicted starting time's upper bound and the realized starting time, and we let w_i the reciprocal of real execution time et^r_{ijk} of task t_i on VM vm_{jk}, i.e., $w_i = 1/et^r_{ijk}$.

Apart from the guarantee ratio and stability, the total energy consumption for executing a set of tasks T is also an important metric to evaluate the performance of a cloud data center. So, in this paper, we also focus on minimizing the total energy consumption, which can be represented as follows (Beloglazov et al., 2012).

$$\text{Minimize} \sum_{j=1}^{n} \int_{st}^{et} (k \cdot p_j \cdot y_j^t + (1-k) \cdot p_j \cdot u(t)) dt. \tag{15}$$

where n is the number of hosts in the system; st and et are the start time and the end time of executing the task set T, respectively; $y_j^t \in \{1, 0\}$ denotes whether host h_j is active at time instant t; k is the fraction of energy consumption rate consumed by the idle host (e.g., 70%); p_j is the host overall power consumption; $u(t)$ is the CPU utilization of host h_j at time instant t.

4. Algorithm design

In this section, we present our algorithm for real-time task and computing resource scheduling under uncertain cloud computing environments. The algorithm incorporates both the proactive and the reactive scheduling methods. The proactive scheduling is used to build baseline schedules based on redundancy, where a protective time cushion between tasks' finish time lower bounds and their deadlines is added to guarantee task deadlines. The reactive scheduling is dynamically triggered to generate proactive baseline schedules in order to account for various disruptions during the course of executions (e.g., urgent tasks arrive, tasks just have been finished, system's workload become too heavy, and the like). In addition, we propose three strategies to scale up and down the computing resources according to system workload.

We treat the following five events as disruptions: (1) a new task arrives; (2) the system becomes overloaded; (3) a new urgent task arrives or tasks waiting in WQ become urgent; (4) a VM finishes a task; and (5) some VMs' idle time exceeds the preestablished threshold I_u, as demonstrated with a system execution timing diagram shown in Fig. 2. These five disruptions take place discretionarily, and arbitrarily; if any of the disruptions occurs, the corresponding reactive scheduling will be triggered.

To facilitate the presentation of our scheduling strategies. Our scheduling rules are given below.

Rule 1. Each virtual machine can only execute one task at any time instant.

Rule 2. When a task t_i cannot be finished in worst-case on any VMs before its deadline (i.e., $ft_{ijk}^{+} > d_i$, $\forall vm_{jk}$), we still schedule it to a certain VM if its finish time's lower bound is not larger than its deadline on some VM (i.e., $ft_{ijk}^{-} \leq d_i$, $\exists vm_{jk}$).

Rule 3. An urgent task is allowed to replace the non-urgent waiting task as a new waiting task on a certain VM, and the replaced non-urgent waiting task will be returned to WQ. Fig. 3(a) illustrates an example of property 3.

In Fig. 3(a), the task t_2 is waiting in the VM vm_{11}, thus, wtk_{11} equals to t_2. As the waiting task t_2's laxity L_2 is 200 that is larger than $L_d = 120$, and the task t_3's laxity L_3 is 80 that is less than $L_d = 120$, t_2 is urgent and t_3 is non-urgent. According to Rule 3, the urgent task t_3 is allowed to replace the non-urgent task t_2 to be a new waiting task on VM vm_{11}, and task t_2 is returned from the local queue (LQ) of VM vm_{11} to the waiting queue (WQ). Rule 3 makes urgent tasks start earlier, increasing possibility of meeting the urgent tasks' timing requirements.

Rule 4. Task transfer is overlapped with the task execution on the same VM. Therefore, a task transfer time is hidden and has no impact on the starting time of the task.

Rule 5. The waiting task on a VM is allowed to start as soon as the executing task on the same VM has finished.

Fig. 3(b) shows an example of property 4 and property 5. In Fig. 2(b), tt_{ijk} represents the transmission time of task t_i to VM vm_{jk}, e.g., t_{211} represents the transmission time of task t_2 to VM vm_{11}. As this figure shows that the transmission time t_{211} of task t_2 to VM vm_{11} is overlapped with the execution of task t_1, and task t_2 can be executed as soon as task t_1 is just finished. Therefore, the time cushion (time$= ft_{111}^{+} - ft_{111}^{r}$) is removed and the resource utilization of VM vm_{11} is improved.

The PRS performs the following operations when a new task arrives, as shown in Algorithm 1.

Algorithm 1. PRS – On the arrival of new tasks

```
1:  WQ ← NULL, UTQ ← NULL;
2:  for each new task t_i do
3:    if L_i < L_d then
4:      UTQ ← t_i;
5:    else
6:      WQ ← t_i;
7:      Sort all the tasks in the WQ by their laxity L_i in a non-descending order;
8:      if inequality (5) comes into existence then
9:        vm_jk ← ScaleUpComputingResource();
10:       if vm_jk ! = NULL then
11:         Move task t_i from WQ to VM vm_jk as executing task;
12:         vm_jk ← SearchWaitingTask();
13:       end if
14:     end if
15:   end if
16: end for
```

When a new task t_i arrives, algorithm **PRS** will check whether this task is urgent. If this task is urgent, it will be delivered to the urgent task queue (UTQ) (Lines 3–4). Otherwise, task t_i will be added to the waiting queue (WQ) (Line 6) and then all the waiting tasks in WQ are sorted by their laxity in an ascending order (Line 7). After that, algorithm **PRS** will estimate whether the requirements of waiting tasks in WQ exceed the computing resources in the system, i.e., whether inequality (5) comes into existence (Line 8). If inequality (5) is true, the function **ScaleUpComputingResource()** is called to add a new VM vm_{jk} for the new arrival task t_i (Line 9). After task t_i has been scheduled to VM vm_{jk} as an executing task (Line 11), the function **SearchWaitingTask()** is called to search a task from WQ to this new VM as a waiting task (Line 12).

Algorithm 2. **Function** ScaleUpComputingResource()

```
1:  hostList ← all the active hosts;
2:  Select a VM vm_jk with minimal MIPS that can finish
    task t_i within its deadline considering the delay of
    creating vm_jk;
3:  if vm_jk ! = NULL then
4:    for each host h_j in hostList do
5:      if VM vm_jk can be created on host h_j then
6:        Create VM vm_jk on host h_j; return vm_jk;
7:      end if
8:    end for
9:  end if
10: Select a VM vm_jk with minimal MIPS that can finish
    task t_i within its deadline considering the delays of
    turning on a host and creating VM vm_jk;
11: if vm_jk ! = NULL then
12:   Turn on a host h_j and then create VM vm_jk on it;
13:   hostList ← h_j; return vm_jk;
14: end if
```

In the function **ScaleUpComputingResource()**, as shown in Algorithm 2, we propose two strategies to add a new VM on a certain physical host. The strategy one is creating a new VM on original host that meets the timing requirement of the corresponding tasks (Lines 2–9). If the strategy one is infeasible, the strategy two turns on a new host and creates a VM on this host (Lines 10–14).

Let st_{h_j} and $at_{vm_{jk}}$ be the startup time of host h_j and the creation time of VM vm_{jk}, respectively.

When task t_i is scheduled to a new VM vm_{jk}, the start time $\widetilde{st}_{ijk}$ of task t_i on VM vm_{jk} is determined by how vm_{jk} was created, which can be described as follows.

$$\widetilde{st}_{ijk} = \begin{cases} ct + at_{vm_{jk}}, & \text{if strategy one,} \\ ct + st_{h_j} + at_{vm_{jk}}, & \text{if strategy two.} \end{cases} \quad (16)$$

Algorithm 3. Function SearchWaitingTask() – On the reaction to completion of a task or creation of a VM

```
1:  t_i ← get the task at the head in WQ;
2:  while t_i != NULL do
3:    Calculate the start time st~_ijk execution time et~_ijk and
      finish time ft~_ijk of task t_i waiting on VM vm_jk;
4:    if ft+_ijk ≤ d_i then
5:      Move task t_i from WQ to VM vm_jk as a waiting task;
6:      break;
7:    else
8:      t_i ← get the next task in WQ;
9:    end if
10: end while
```

In function **SearchWaitingTask()**, as given in Algorithm 3, the task with low laxity will be considered first (Line 1 and Line 8) together with the proactive method (Lines 4) to guarantee the timing constraints. In addition, as the finish time of a task on a VM is uncertain, we regard the completion of a task by a VM as an event. When this event occurs, if there exists a waiting task on the VM, the waiting task starts to execute immediately, as shown in Property 5; this event also triggers the operation of finding a new waiting task for the VM from the waiting queue, which is performed by the function **SearchWaitingTask()**.

When a new urgent task arrives or some tasks waiting in WQ become urgent as shown in Fig. 2 at times 25, 38 and 46, these tasks are delivered to the urgent task queue (UTQ), and the function **ScheduleUrgentTasks()** is triggered to schedule the urgent tasks in UTQ.

Algorithm 4. Function ScheduleUrgentTasks() – On the occurrence of urgent tasks

```
1:  for each urgent task t_i in UTQ do
2:    minFinishTimeUpper ← +∞, vmUpper ← +∞;
3:    minFinishTimeLower ← +∞, vmLower ← +∞;
4:    for each VM vm_jk in the system do
5:      if etk_jk == NULL || wtk_jk == NULL || wtk_jk is
      non-urgent then
6:        Calculate the start time st~_ijk execution time et~_ijk
        and finish time ft~_ijk of task t_i on VM vm_jk;
7:        if ft+_ijk ≤ d_i & ft+_ijk ≤ minFinishTimeUpper then
8:          minFinishTimeUpper ← ft+_ijk; vmUpper ← vm_jk;
9:        else if vmUpper == NULL
        & ft-_ijk ≤ minFinishTimeLower then
10:         minFinishTimeLower ← ft-_ijk; vmLower ← vm_jk;
11:       end if
12:     end if
13:   end for
14:   if vmUpper != NULL then
15:     Move task t_i from UTQ to VM vmUpper as
      executing or waiting task;
16:   else
17:     vm_jk ← ScaleUpComputingResource();
18:     if vm_jk != NULL then
19:       Move task t_i from UTQ to VM vm_jk as executing
        task;
20:       vm_jk ← SearchWaitingTask();
21:     else if vmLower != NULL then
22:       Move task t_i from UTQ to VM vmLower as
        executing or waiting task;
23:     else
24:       Reject task t_i;
25:     end if
26:   end if
27: end for
```

In function **ScheduleUrgentTasks()**, as shown in Algorithm 4, we employ three policies to schedule an urgent task to a VM. In policy one, three kinds of initiated VMs (i.e., a VM's executing task etk_{jk} is null, a VM's waiting task wtk_{jk} is null, a VM's waiting task wtk_{jk} is non-urgent) are considered for this urgent task (Line 5), and the initiated VM that can finish the urgent task with the minimum earliest finish time is selected for this urgent task (Lines 7–8). If the first policy can select a VM for this urgent task, it will be scheduled to the selected VM (Lines 14–15). If the first policy is infeasible, function **ScaleUpComputingResource()** will be called to scale up VM for this urgent task (Line 17). Besides, if the second policy is feasible and the urgent task has been scheduled to the new VM as an executing task (Line 19), then the function **SearchWaitingTask()** will be called to search a task from WQ for this new VM as a waiting task (Line 20). If the above two policies are infeasible, but the finish time's lower bound of this urgent task on some initiated VMs is not greater than its deadline (Lines 9–10), we schedule it to the VM with the minimum earliest finish time (Lines 21–22). If all the above three policies cannot schedule this task, it will be rejected (Line 24).

Theorem.1. *The time complexity for scheduling a task set T with algorithm PRS is $O(|T|N_{wt}log(N_{wt}) + |T|N_{vm})$, where $|T|$ represents the count of tasks in T, N_{wt} is the count of waiting tasks in WQ, N_{vm} is the count of initiated VMs.*

Proof. It takes $O(N_{wt}log(N_{wt}))$ to sort all the waiting tasks in WQ (Line 7, Algorithm 1). In Algorithm 2, it takes $O(N_a)$ to check if a new VM can be created on an active host directly (Lines 4–8, Algorithm 2), where N_a is the count of active hosts in the system. It takes $O(N_a)$ to add a new VM by turning on a shut host (Lines 11–14, Algorithm 2). Thus, the time complexity of function ScaleUpComputingResource() is $O(N_a)$. In addition, the time complexity of function SearchWaitingTask() is $O(N_{wt})$ (Algorithm 3). Thus, the time complexity of Algorithm 1 is $O(N_{wt}log(N_{wt}) + N_a + N_{wt}) = O(N_{wt}log(N_{wt}) + N_a)$. In function ScheduleUrgentTask(), it takes $O(N_{vm})$ to find a feasible VM for a urgent task (Lines 4–13, Algorithm 4). In addition, the time complexity of function ScaleUpComputingResource() and SearchWaitingTask() are $O(N_a)$ (Line 17, Algorithm 4) and $O(N_{wt})$ (Line 20, Algorithm 4), respectively. The time complexity of scheduling an urgent task by Algorithm 4 is $O(N_{vm} + N_a + N_{wt})$. Based on the aforementioned analysis, the time complexity for scheduling a task set T is calculated as $O(|T|)max\{O(N_{wt}log(N_{wt}) + N_a), O(N_{vm} + N_a + N_{wt})\} = O(|T|N_{wt}log(N_{wt}) + |T|N_{vm})$, since $N_a \le N_{vm}$. □

If the workload of the system become light and there exist some VMs whose idle times are larger than the idle upper threshold I_u, as shown in Fig. 2 at time 28 or 52, then function **scaleDownComputingResource()**, as shown in Algorithm 5, will be called to scale down the computing resources by turning off idle VMs and physical hosts. Firstly, it turns off those VMs whose idle time exceeds I_u (Lines 1–5). Then it separates all the idle hosts and active hosts (Line 6), and sorts all the active hosts by their CPU capacity utilized in a non-descending order (Line 7). After these operations, all the initiated VMs on active physical hosts will be consolidated to a minimal number of hosts (Lines 8–13), and all the idle hosts can be finally turned off (Lines 14–16) to reduce energy consumption.

Theorem.2. *The time complexity of function ScaleDownComputingResource() is $O(N_a^2 + N_{vm})$, where N_a is the count of active hosts and N_{vm} is the count of VMs before scaling down the system's computing resources.*

Algorithm 5. **Function** ScaleDownComputingResource()

```
1:  for each idle VM vm_jk in the system do
2:    if vm_jk's idle time exceeds I_u then
3:      Delete VM vm_jk;
4:    end if
5:  end for
6:  idleHostList← all the idle hosts; activeHostList← all the active hosts;
7:  Sort activeHostList by CPU capacity utilized in a non-descending order;
8:  for each host h_j in activeHostList do
9:    if all the working VMs on host h_j can be migrated to other active hosts then
10:     Migrate all the working VMs on host h_j to corresponding hosts;
11:     Move host h_j from activeHostList to idleHostList;
12:   end if
13: end for
14: for each host h_j in idleHostList do
15:   Delete all the VMs on host h_j, then turn off host h_j;
16: end for
```

Proof. The time complexity of deleting all the idle VMs whose idle time exceeds the preestablished upper threshold is $O(N_{vm})$ (Lines 1–5, Algorithm 5). It takes $O(N_a log(N_a))$ to sort all the active hosts by their CPU capacity utilized in a non-descending order (Line 7, Algorithm 5). The time complexity of reallocating the initiated VMs in the system is $O(N_a|VM_j|N_a)$ (Lines 8–13, Algorithm 5). Since the $|VM_j|$ is a value that is not greater than 12, $O(N_a|VM_j|N_a) = O(N_a^2)$. It takes $O(N_i)$, where N_i is the count of idle hosts in the system, to delete all the idle hosts in the system (Lines 14–16, Algorithm 5). Therefore, the complexity of function ScaleDownComputingResource() is $O(N_{vm} + N_a \log(N_a) + N_a^2 + N_i)$. Since N_i is less than N_a, $O(N_{vm} + N_a \log(N_a) + N_a^2 + N_i) = O(N_a^2 + N_{vm})$. □

5. Performance evaluation

To demonstrate the performance improvements gained by PRS, we quantitatively compare it with a baseline algorithm non-Migration-PRS (NMPRS in short) and three existing algorithms – Earliest Deadline First (EDF) (Mills and Anderson, 2010), minimum completion time (MCT) (Li et al., 2011) and complete rescheduling (CRS) (Van de Vonder et al., 2007). The brief explanation of these algorithms and the motivation for selecting them as competing algorithms are as follows.

NMPRS: NMPRS does not employ the VM migration strategies while scaling down the computing resources, (See Lines 7–13, Algorithm 5). Through comparing with algorithm NMPRS, the effectiveness of the VM migration strategies in PRS can be tested.

EDF: all the tasks are arranged on VMs by their deadlines in a non-descending order while guaranteeing the timing constraints of these tasks. In order to guarantee the deadlines of tasks, this algorithm utilizes the worst-case execution time of tasks during scheduling, and all the waiting tasks will be executed exactly as the baseline schedule.

It is noting that EDF is an optimal scheduling algorithm for independent real-time tasks on preemptive uniprocessors, the goal of comparing PRS with EDF is to demonstrate that PRS can produce better scheduling performance for real-time tasks with uncertain execution time.

MCT: MCT maps a new task on a VM that can complete this task in the earliest time while maintaining the finish time upper bound before the task's deadline. In addition, all the tasks are allocated to VMs upon their arrivals.

MCT is a widely used greedy algorithm and Li et al. (2011) have proved that MCT has the best performance under uncertain environment comparing with another two greedy algorithms (i.e., Min–min and Max–min). Therefore, we select MCT as the representative of classic greedy scheduling algorithms to demonstrate the performance improvements gained by PRS comparing with classic greedy scheduling algorithms.

CRS: when new tasks arrive, CRS will completely reschedule the new tasks and all the waiting tasks in the system. Notice that accurate task execution times are assumed to be available before scheduling in this algorithm, and schedule results of CRS can be considered as near-optimal because the schedule is reoptimized at any decision point by fully using all information available at that time (Herroelen and Leus, 2005).

However, the high time complexity of this algorithm limits its application in real-time cloud system (Van de Vonder et al., 2007). Through comparing PRS with CRS, we can analyze how near the performance between PRS and near-optimal algorithm.

In order to compare the efficiency of the algorithms, we utilize the following performance metrics to evaluate their performances.

(1) Guarantee Ratio: the ratio of tasks finished before their deadlines;
(2) Resource Utilization: the average host utilization, which can be calculated as: $RU = \left(\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{|VM_j|} l_i^r \cdot o_{ijk}\right) / \left(\sum_{j=1}^{n} c_j \cdot wt_j\right)$, where l_i^r is the realized length of task t_i, and wt_j is the active time for host h_j during an experiment.
(3) Total Energy Consumption: the total energy consumed by the hosts in a cloud data center for executing a task set T.
(4) Stability: the weighted sum of the absolute deviations between the predicted starting time of tasks in the baseline schedule and their realized starting time during actual schedule execution.

5.1. Experiment on a real cluster

In this experiment, we deploy an experimental cloud environment using Apache CloudStack 4.2.0. We set up a KVM cluster with five Dell Optiplex 7010 MT, each of physical host has a CPU (i3 3.9 GHz 4 cores), 3.7 G memory and 500 G disk storage, and the peak power of a host is 200 W. In addition, the CPU and memory required for each VM are two CPU cores (i.e., 2×3.9 GHz) and 1.5 G, respectively. Besides, the five machines run on a 1 Gbps Ethernet network.

Two applications are selected to perform our evaluation in this subsection. The first application is to compute an approximation of π, and it is called π-App (Hagimont et al., 2013). For the second application, we utilize CloudSim that performs aforementioned algorithms as a real runnable application, which is named *CloudSim*-App.

Firstly, we want to verify that the performance of VMs subjects to degradation or uncertainties when multiple VMs share the hardware resources of a physical host. In the experiment, two VMs with the same parameters are collocated in a physical host. Two execution conditions are tested. In Condition 0, only one VM executes an application and the other VM is idle, while in Condition 1 the two VMs execute simultaneously. Table 1 shows the sampled execution times (in ms) of the applications π-App and *CloudSim*-App at the two different conditions. For each case, 15 execution times are sampled.

For each sample data set in Table 1, we can obtain its mean (denoted by $\bar{x}$) and variance (denoted by s^2) values. For application π-App, the sample means of the two conditions are $\bar{x}_0 = 8333.4$ and $\bar{x}_1 = 8372.8$; their sample variance are $s_0^2 = 1.2571$ and $s_1^2 = 273.46$.

To show that change of execution condition will result in uncertain execution time, we investigate two hypotheses: the null hypothesis that the execution times of the two execution conditions are the same, and the alternative hypothesis that the two execution times are different. The null and alternative hypotheses

Table 1
Execution times for applications in different conditions.

Application	π-App	*CloudSim*-App
Condition 0	8334 8334 8334 8332 8334 8334 8333 8333 8334 8330 8334 8333 8334 8334 8334	45555 45806 46230 47448 46921 46246 47921 46136 47061 46240 46621 46530 45723 46666 45729
Condition 1	8383 8381 8380 8380 8376 8357 8330 8344 8387 8386 8384 8382 8379 8373 8370	61157 61211 61426 59013 59102 61137 58514 56830 58660 58673 59071 57939 59211 59254 59366

Table 2
Performance comparison in real cluster.

Metric	Algorithm			
	PRS	NMPRS	EDF	MCT
Guarantee ratio	95.23%	95.82%	91.25%	90.93%
Resource utilization	0.8837	0.4382	0.7742	0.5476
Total energy($\times 10^7 J$)	0.7328	1.0325	0.8423	0.9504
Stability	0.1154	0.1097	0.8527	0.2031

for the two-tailed test for π-App are written as follows (Anderson et al., 2013).

$$H_0 : \mu_0 = \mu_1 \text{ vs } H_1 : \mu_0 \neq \mu_1. \tag{17}$$

The test statistic t for above hypothesis tests can be calculated as follows.

$$|t| = \frac{|\bar{x}_0 - \bar{x}_1|}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}} = \frac{39.4}{4.3} = 9.2, \tag{18}$$

where n_0 and n_1 are the sample size; $n_0 = n_1 = 15$.

Using a significance level of $\alpha = 0.01$ and $(n_0 + n_1 - 2)$ degrees of freedom, the t distribution can be calculated as $t_{(1-\alpha/2)}(n_0 + n_1 - 2) = t_{0.995}(28) = 2.763$. Comparing the computed value of $|t| = 9.2$ with $t_{0.995}(28) = 2.763$, it can be concluded that the null hypothesis is clearly rejected, i.e., there exists difference between the execution times of application π-App from the two different execution conditions. This difference is more noticeable for *CloudSim*-App where $|t| = 52.17$ and $t_{0.995}(28) = 2.763$. As a result, there is overwhelming evidence that the performance of VMs subjects to degradation or uncertainties when multiple VMs share the hardware resources of a physical host.

According to the execution times of π-App and *CloudSim*-App in Table 1, we assume their execution times to be $\widetilde{et}_{ijk} = [8.2, 8.4]$s and $\widetilde{et}_{ijk} = [45, 62]$s, respectively. To emulate the diverse execution times of real-time tasks, we control the repeat times (rt in short) of π-App and *CloudSim*-App, and calculate the execution times of tasks with $rt \times \widetilde{et}_{ijk}$. In this experiment, we utilize π-App to produce a task set, denoted as $T_1 = \{t_i, i = 1, b, \cdots, 10\}$, where t_i means a task that repeats π-App i times, and the execution time of task t_i is expressed as $\widetilde{et}_{ijk} = i \times [8.2, 8.4]$s. Similarly, *CloudSim*-App is used to form another task set, denoted as $T_2 = \{t_i, i = 11, 12, \cdots, 20\}$, where t_i means a task that repeats *CloudSim*-App $(i - 10)$ times, and the execution time of task t_i is expressed as $\widetilde{et}_{ijk} = (i - 10) \times [45, 62]$s.

To realize the dynamic nature of real-time tasks in cloud computing environment, the tasks in task set T_1 and T_2 are selected randomly after a time interval and submitted to scheduler. The time interval between two consecutive tasks is a variable, and let it be uniformly distributed among 30 s and 120 s. In addition, the deadline of each task is calculated as $d_i = a_i + \widetilde{et}^{+}_{ijk} + U(0, 600)$ s, where $U(0, 600)$ s denotes a variable that subjects to uniformly distributed among 0 s and 600 s. The task count for evaluating the four algorithms (i.e., PRS, NMPRS, EDF and MCT) is 500, and the algorithm CRS is not implemented since the accurate task execution time cannot be gained in real environment. Table 2 shows the performance comparison for the four algorithms in this experimental cloud environment.

As shown in Table 2, the guarantee ratios of algorithm PRS, NMPRS, EDF and MCT are 95.23%, 95.82%, 91.25% and 90.93%, respectively. The reason for the higher guarantee ratios for algorithm PRS and NMPRS is that these two algorithms employ strategies to control the impact of uncertainties. It is obvious that PRS and EDF are efficient in the context of the resource utilization, this is because these two algorithms utilize VM migration policies when scaling down computing resources. This result indicates that our strategies for scaling up and down computing resources are effective in practice. The energy consumed by algorithm NMPRS and MCT is higher than that from other two algorithms, which can be due to that their resource utilizations are low. Furthermore, PRS and NMPRS outperform EDF and MCT in terms of stability, this is because PRS and NMPRS employ strategies to prohibit the propagation of uncertainties.

5.2. Simulation experiments

In order to ensure the repeatability of the experiments, we choose the way of simulation to evaluate the performance of aforementioned algorithms. The CloudSim toolkit (Calheiros et al., 2011) is chosen as a simulation platform, and we add some new settings to conduct our experiments, which is similar to (Beloglazov et al., 2012).

Since most data centers provide enough capacity to handle their peak utilization while the average usage is much lower (Burge et al., 2007), we assume that the number of hosts in a cloud data center is infinite. Each host is modeled to have one CPU core with the performance equivalent to 1000, 2000, and 3000 MIPS, 8 GB of RAM and 1 TB of storage, and the peak power of the three different hosts are 250 W, 300 W or 400 W, respectively[2]. Besides, each VM requires one CPU core with 250, 500, 750, 1000 MIPS, 128 MB of RAM and 1 GB of storage (Beloglazov and Buyya, 2012). Each VM carries out a task with variable CPU performance, and only upper and lower bounds of CPU performance are available before scheduling via the technique in Beloglazov and Buyya (2012). In addition, the start-up time of a host is 90 s and the creation time of a VM is 15 s, and the migration time of a VM is determined by its RAM capacity and the bandwidth of the system, which is similar to the model described in Hermenier et al. (2009).

In addition, we let $U[a, b]$ be an uniformly distributed random variable between a and b.

We employ parameter *deadlineBase* to control a task's deadline that can be calculated as.

$$d_i = a_i + U[deadlineBase, a \times deadlineBase]. \tag{19}$$

where parameter *deadlineBase* determines whether the deadline of a task is loose or not. In this paper, we set the value of a is $a = 4$.

[2] http://www.google.com.hk/search?q=Energy+Star+computer+server+qualified+product+list.

Table 3
Parameters for simulation studies.

Parameter	Value (*Fixed*) – (*Varied*)
Task count (10^4)	(2.0)-(1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0)
deadlineBase (s)	(200)-(100,150,200,250,300,350,400)
intervalTime (s)	([0,5])
taskLength (10^5 MI)	([1,2])
taskUncertainty	(0.2)-(0.0,0.05,0.10,0.15,0.20,0.25,0.30,0.35,0.40)
vmUncertainty	(0.2)-(0.0,0.05,0.10,0.15,0.20,0.25,0.30,0.35,0.40)

The parameter *intervalTime* is used to determine the time interval between two consecutive tasks, and it is assumed to be uniformly distributed among 0 and 5.

The parameters *taskUncertainty* and *vmUncertainty* represent the uncertainty upper bounds of tasks and VMs in the system, respectively, and the lower and upper bonds of task t_i's computing length and a VM's performance are modeled as follows.

$$\begin{aligned} l_i^- &= U[1\times 10^5, 2\times 10^5];\\ l_i^+ &= l_i^- \times (1 + U[0, taskUncertainty]);\\ c_{jk}^- &= c_{jk}^+ \times (1 - U[0, vmUncertainty]). \end{aligned} \tag{20}$$

where c_{jk}^+ is the CPU performance capacity required for VM vm_{jk}.

In this experiment, we calculate the realized finish time for a task as follows.

$$ft_{ijk}^r = ft_{ijk}^- + (ft_{ijk}^+ - ft_{ijk}^-) \times U[0, 1]. \tag{21}$$

Note that the parameter ft_{ijk}^r is not available before scheduling, except in algorithm CRS. The values of these parameters are listed in Table 3.

5.2.1. Number of tasks

In this group of experiments, we study the impact of the number of tasks on system performance. Fig. 4 illustrates the experimental results of PRS, NMPRS, EDF, MCT and CRS when the number of tasks varies from 10,000 to 80,000 with an increment of 10,000.

In Fig. 4(a), we can see that with the task count increases, the guarantee ratios for algorithm PRS, NMPRS, EDF, MCT and CRS are stable at 94.42%, 94.66%, 91.06%, 88.03% and 97.92%, respectively. This is because there are enough resources in a cloud, thus when the workload of the system becomes overloaded, more computing resources will be scaled up to satisfy the surplus workload. Unfortunately, tasks' timing constraints never be totally guaranteed although there are enough resources. This is due to the fact that the time overheads for scaling up computing resources violate some urgent tasks' timing constraints. In addition, we observe that the guarantee ratio of PRS is close to the highest performance offered by CRS. PRS improves the guarantee ratio of MCT by an average of 7.26%. This performance improvement is made possible, because PRS and NMPRS employ policies to control the uncertainty propagation. In addition, it can be found that the guarantee ratio of PRS is close to that of CRS which is assumed to be the performance upper bound, and on average is higher than that of EDF and MCT by 3.70% and 7.26%, respectively. This can be explained that PRS and NMPRS employ policies to control the propagation of uncertainties and to take a risk of wasting computing resources by allocating urgent tasks that only lower bounds of their finish time are not greater than their deadlines, to VMs.

Fig. 4(b) shows that the resource utilization of these five algorithms can be roughly classified into three categories. CRS and PRS are the best performer, while NMPRS and MCT the worst. This can be attributed that these algorithms employ different VM relocation policies when scaling down computing resources. The CRS, PRS and EDF employ the VM migration policies that can make the host computing resources utilized efficiently. Besides, CRS and PRS significantly outperform EDF with respect to resource utilization. This is because EDF does not employ the early start policy to execute waiting tasks, which incurs delay (i.e., the time between task's realized finish time and the next task's start time) between two adjacent tasks on the same VM. On average, PRS outperforms NMPRS, EDF and MCT by 14.98%, 7.22% and 17.25%, respectively.

Fig. 4(c) reveals that the total energy consumptions of the four tested algorithms are linearly increases with the number of tasks. This is because the guarantee ratios of these four algorithms vary slightly around different constants; the total tasks' computation lengths are linear to the number of tasks and the total energy consumption of the system is almost linear to the total computation length. We observe that the total energy consumptions of CRS, PRS and MCT are similar, whereas NMPRS consumes the most energy in each test case. NMPRS is not energy efficient, because NMPRS does not adopt the VM migration strategy to consolidate VMs. The low energy consumption of MCT is attributed to its low guarantee ratio.

Fig. 4(d) shows that the stability of PRS, NMPRS, EDF, MCT and CRS are 0.1078, 0.1077, 0.8579, 0.3021 and 0.4577, respectively. Increase of these five algorithms is that the increase of task count seldom affects the baseline schedule. It is not a surprise that the stability of EDF is high (0.8579) since all the waiting tasks on VMs are sorted by their deadlines, such that scheduling a new task to a VM will update the start time of these tasks whose deadlines are not less than the new task's deadline on the same VM. The reason for the high variation of CRS can be contributed to the fact that CRS will completely rescheduling all the waiting tasks and new tasks when new tasks arrive. The high variation of MCT is because MCT schedules all the tasks to VMs as soon as they arrive, thus resulting in the propagation of uncertainties. Finally we see that the stability of PRS and NMPRS always keep at a better level, which demonstrates the efficiency of our strategies in term of controlling the uncertainties while scheduling.

5.2.2. Task deadlines

Fig. 5 shows the impacts of deadlines on the performance of our proposed PRS and NMPRS as well as the existing algorithms – EDF, MCT and CRS.

We observe from Fig. 5(a) that the guarantee ratios of the five algorithms increase correspondingly with the increase of *deadlineBase* (i.e., task deadline becomes looser). This can be interpreted that as the deadlines of tasks are prolonged, the need of scaling up computing resources, hence the related time overhead, is reduced, therefore the tasks' timing constraints become weaker. In addition, Fig. 5(a) shows that PRS and NMPRS have higher guarantee ratios than EDF and MCT. This is due to the following three reasons. First of all, PRS and NMPRS employ reactive strategies, which gives urgent tasks high priority so that more tasks can meet their deadline; therefore, they are better than MCT. Secondly, PRS and NMPRS use start early policy to eliminate the waste of time cushion between two adjacent tasks on the same VM, so they perform better than EDF. Thirdly, PRS and NMPRS can efficiently control the propagation of uncertainties by limiting the waiting tasks on VMs, therefore they are better than EDF and MCT on guarantee ratio. When the *deadlineBase* is not greater than 200s, PRS guarantees more tasks' timing constraints than EDF and MCT by, on average, 10.59% and 12.87%, respectively.

From Fig. 5(b), we can see that when *deadlineBase* increases, the resource utilization of the five algorithms increase. This can be contributed to the fact that as the deadlines of tasks become looser, more tasks can be finished in the current active hosts without starting more hosts, thus the utilization of active hosts is higher. When *deadlineBase* is larger than 300 s, the resource utilization of PRS outperforms NMPRS, EDF and MCT by 18.55%, 8.94% and 18.43%,

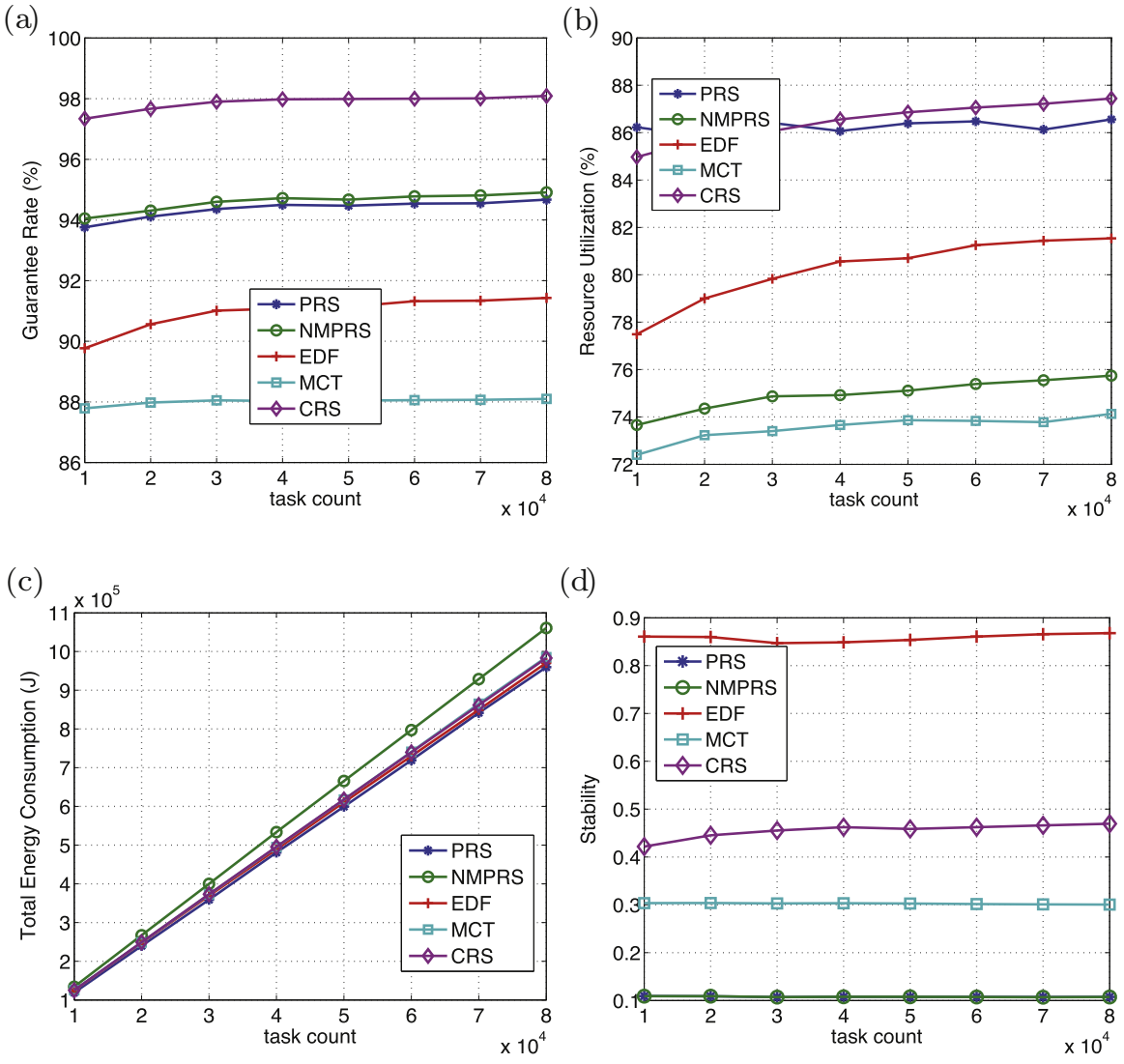


Fig. 4. Performance impact of task number.

respectively. The explanation for this experimental result is similar to that in Fig. 4(b).

Fig. 5(c) shows that the total energy consumption of PRS, NMPRS, EDF, MCT and CRS become larger with the increase of *deadlineBase*. This is because that as the *deadlineBase* increases, more tasks will be executed, thus more hosts and VMs need to work longer, resulting in more energy being consumed. When *deadlineBase* is less than 200 s, the total energy consumption of PRS is higher than that of EDF and MCT. This may be contributed to the following two reasons. Firstly, PRS completes more tasks than the other algorithms. Secondly, in order to complete more tasks, PRS will schedule some tasks that cannot be completed before their deadlines to VMs which dissipates much computing resources. Furthermore, when *deadlineBase* is greater than 250 s, the total energy consumption of PRS is close to CRS.

From Fig. 5(d), we can observe that when the *deadlineBase* increases, the stability of EDF and CRS increases significantly, but that of other algorithms is almost constant. The reason for EDF is that when tasks' deadlines become looser, more tasks can tolerate to wait on the same VMs longer and more urgent tasks will be inserted before them, which results in updating waiting tasks' start time. For CRS, when deadline becomes looser, more tasks can tolerate to be rescheduled more. Besides, the stability of PRS and NMPRS outperform MCT and CRS on average by 192.33% and 298.77%, respectively. The reasons are similar to Fig. 4(d).

5.2.3. Task uncertainty

Fig. 6 illustrates the performance of PRS, NMPRS, EDF, MCT and CRS when the *taskUncertainty* value varies from 0.00 to 0.40 with an increment of 0.05.

Fig. 6(a) shows that the guarantee ratios of the five algorithms descend at different rate as the *taskUncertainty* increases, and this trend is especially outstanding with EDF and MCT. This is because EDF and MCT do not employ any strategies to control the uncertainties while scheduling. Besides, the guarantee ratio of PRS is close to that of CRS and NMPRS, and on average is higher than EDF and MCT by 4.02% and 7.03%, respectively.

Fig. 6(b) reveals that the resource utilization of PRS, NMPRS and EDF descend significantly, but that of CRS and MCT stay at a fixed level. This is due to a few reasons. Firstly, PRS and NMPRS take a risk of dissipating some computing resources to execute those urgent tasks whose deadlines are between their lower and upper finish time. Secondly, for EDF, the interval between tasks' finish

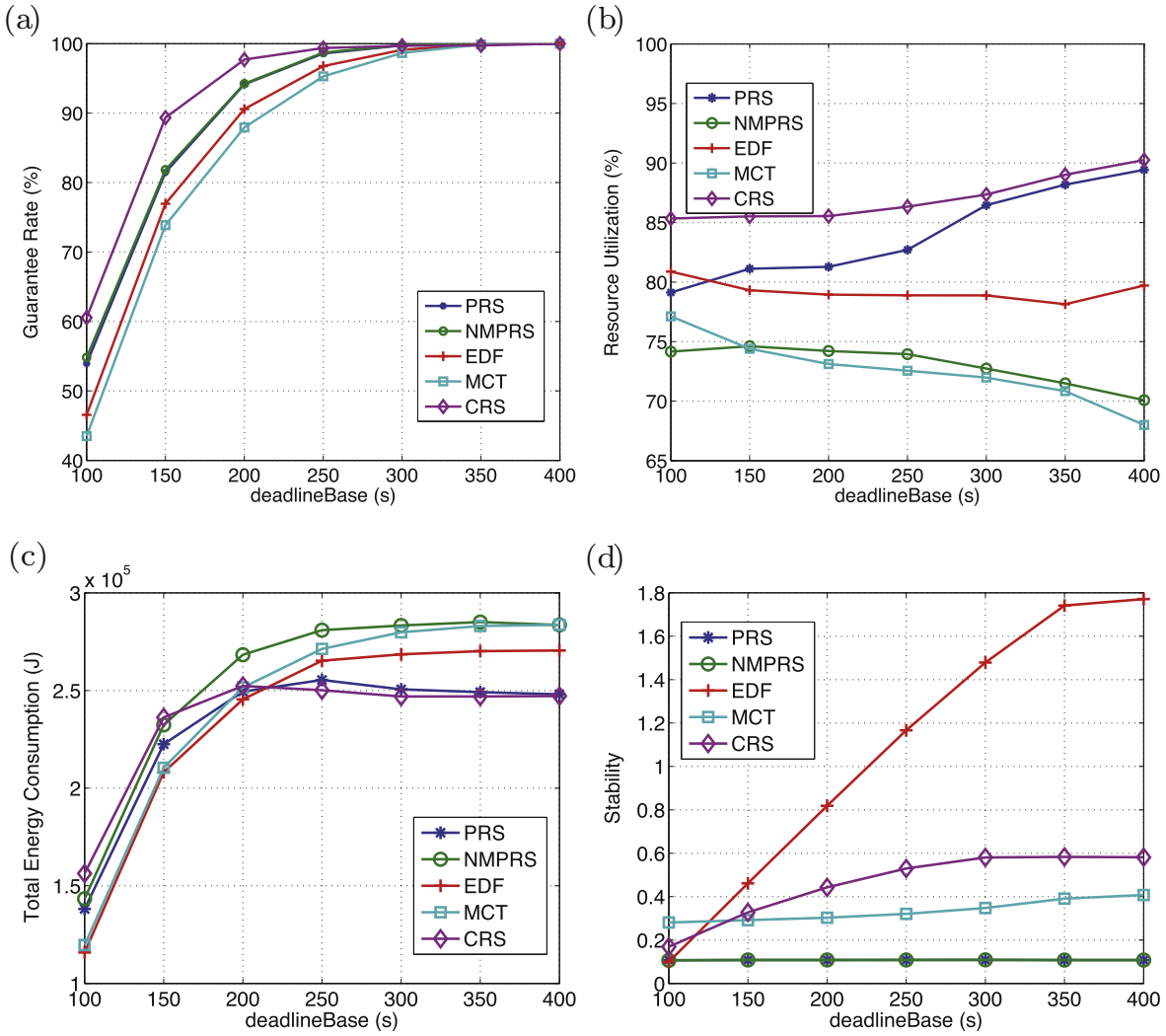


Fig. 5. Performance impacts of task deadlines.

time lower and upper bounds becomes larger as the *taskUncertainty* increases, thus the time cushions wasted by EDF become larger, resulting in lower resource utilization. Thirdly, since the accurate completion time is available in CRS, the resource utilization of CRS can keep stable at a high level. Lastly, MCT only schedules the tasks whose upper completion time is not greater than their deadlines to VMs, and employs the start early policy to eliminate dissipating time cushion, but does not employ VM migration policy when scaling up and down computing resources, thus its resource utilization keeps stable in low level.

Fig. 6(c) depicts that with the increase of *taskUncertainty*, the total energy consumptions of PRS, NMPRS and EDF increase significantly. For algorithm PRS and NMPRS, it is reasonable that as *taskUncertainty* becomes larger, more tasks that cannot be completed before their deadlines are scheduled to VMs, thus the system wastes more computing resources. For EDF, with *taskUncertainty* increases, the time cushion between tasks' predicted and realized finish time becomes larger, thus the idle time of computing resources becomes longer. Besides, the total energy consumptions of MCT and CRS are basically stable. For CRS, its guarantee ratio and resource utilization keep stable. For MCT, its resource utilization is nearly unchanged while its guarantee ratio descends.

Fig. 6(d) shows that when the *taskUncertainty* increases, the stability of PRS, NMPRS and MCT increases. This is because the interval between tasks' finish time lower and upper bounds increases with the *taskUncertainty*, thus the variation between the predicted and realized start time of next tasks becomes larger. In contrary, the stability of EDF decreases as the *taskUncertainty* increases. This is because when *taskUncertainty* increases, the urgent tasks' finish time upper bounds become larger, and less urgent tasks will be accepted. Besides, the stability of PRS is, on average, (184.46%) lower than that of MCT, since MCT does not control the propagation of uncertainties among waiting tasks. Furthermore, the stability of CRS stays almost at a fixed level (about 0.4432). This is because when a new task arrives CRS will completely reschedule all the waiting tasks together with the new task.

5.2.4. Virtual machine uncertainty

We conduct a group of experiments to observe the impact of virtual machine uncertainty on the performance of the four algorithms (see Fig. 7). We vary the *vmUncertainty* value from 0.00 to 0.40 with an increment of 0.05.

From Fig. 7(a) we can see that when *vmUncertainty* increases, the guarantee ratios of the five algorithms descend, especially the trends of EDF and MCT are outstanding. The explanation for this experimental result is similar to that for Fig. 6(a). Furthermore, on average the guarantee ratio of PRS outperforms EDF and MCT by 11.06% and 13.94%, respectively.

Fig. 7(b) reveals that the resource utilization of the five algorithms descend significantly with the increase of *vmUncertainty*. This

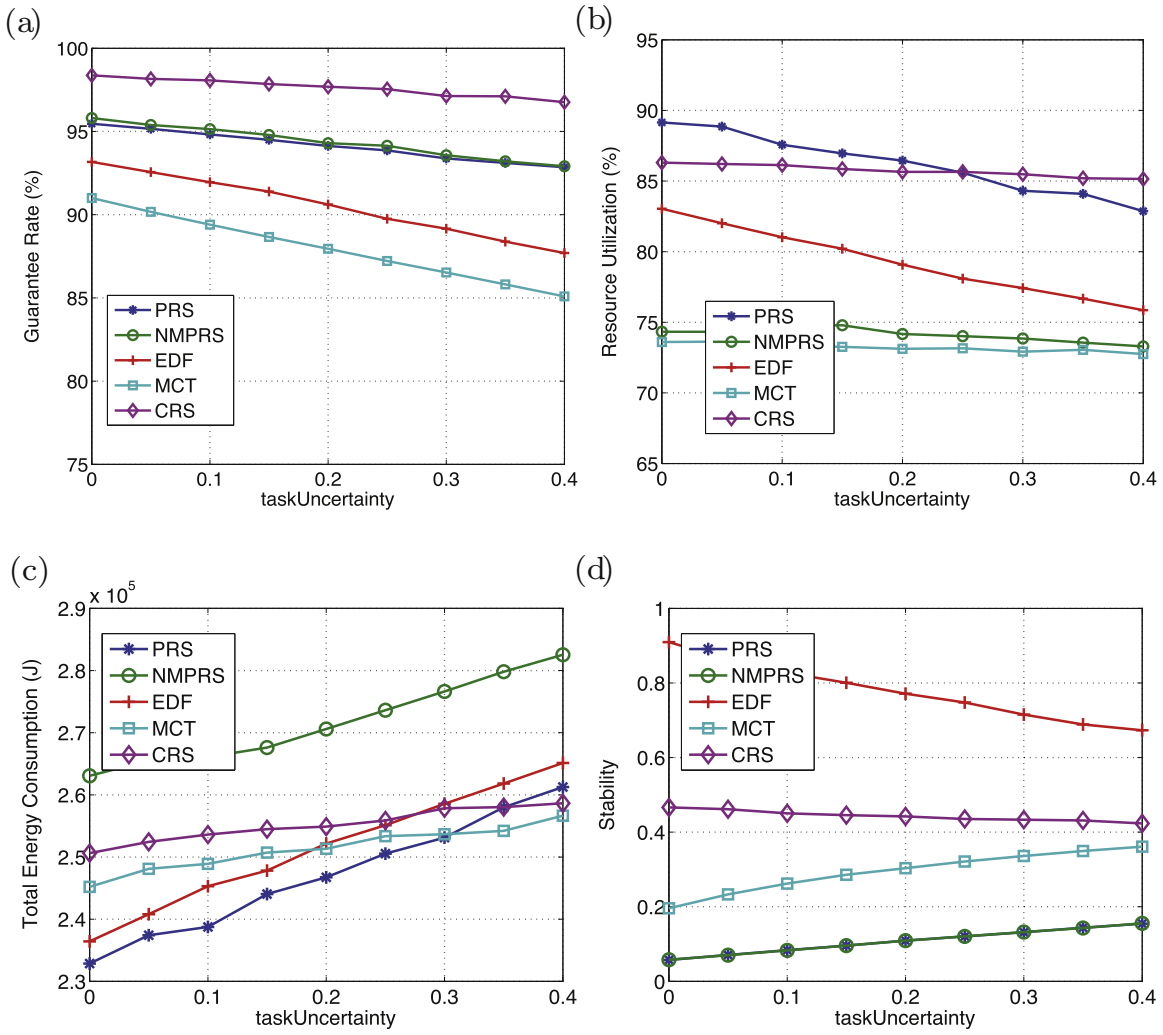


Fig. 6. Performance impact of task uncertainty.

is because as the *vmUncertainty* increases the performance degradation of VMs will become significant, resulting in consuming more resources of physical hosts. The reason of PRS outperforming other algorithms is similar to that for Fig. 4(b).

Fig. 7(c) depicts that with the increase of *vmUncertainty*, the total energy consumption of PRS, NMPRS, EDF and CRS increase. The reason for PRS and NMPRS is that they will allocate more urgent tasks that cannot be completed before their deadlines to VMs as *vmUncertainty* becomes larger, thus costing more resources. For algorithm CRS, its guarantee ratio is basically constant, but its resource utilization decreases significantly with *vmUncertainty*. It is not a surprise that MCT shows an opposite trend as compared to other algorithms, which can be explained with the significant decrease in the guarantee ratio and stable resource utilization of MCT.

Fig. 7(d) shows that the stabilities of PRS, NMPRS and MCT increase when *vmUncertainty* varies from 0.00 to 0.40. In contrast, the stability of EDF decreases from 0.8686 to 0.6697 when *vmUncertainty* varies from 0.00 to 0.40. The explanation for this experimental result is similar to that in Fig. 6(d).

5.2.5. Real-world traces

To further evaluate the practicality and efficiency of our algorithms, in this subsection, we compare these algorithms based on real-world trace which is the latest version of the Google cloud trace log.[3] We choose 955,626 continuous tasks starting from *timestamp* = 1, 468, 890 to *timestamp* = 1, 559, 030. Fig. 8 shows the task distribution over the period.

Since the trace log does not contain definite information about computation length and deadlines of tasks, we set these two parameters of tasks as follows, which is similar to (Moreno et al., 2013).

- Task computation length lower bound l_i^- is calculated based on the execution duration and the average CPU utilization. Since the tracelog does not contain the data of task length in MI, we employ the method proposed in (Moreno et al., 2013) to estimate the task length.

$$l_i^- = (ts_{finish} - ts_{schedule}) \times U_{avg} \times C_{CPU}, \tag{22}$$

where ts_{finish} and $ts_{schedule}$ represent the timestamp of finish and schedule event; U_{avg} denotes the average CPU usage of this task. All the three values can be obtained in the tracelog. C_{CPU} represents the processing capacity of the CPU in Google cloud, because the data of machines' capacity is not included in the trace we assume that it is similar to our experiment settings for hosts, C_{CPU} =1000 MIPS.

[3] http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1.

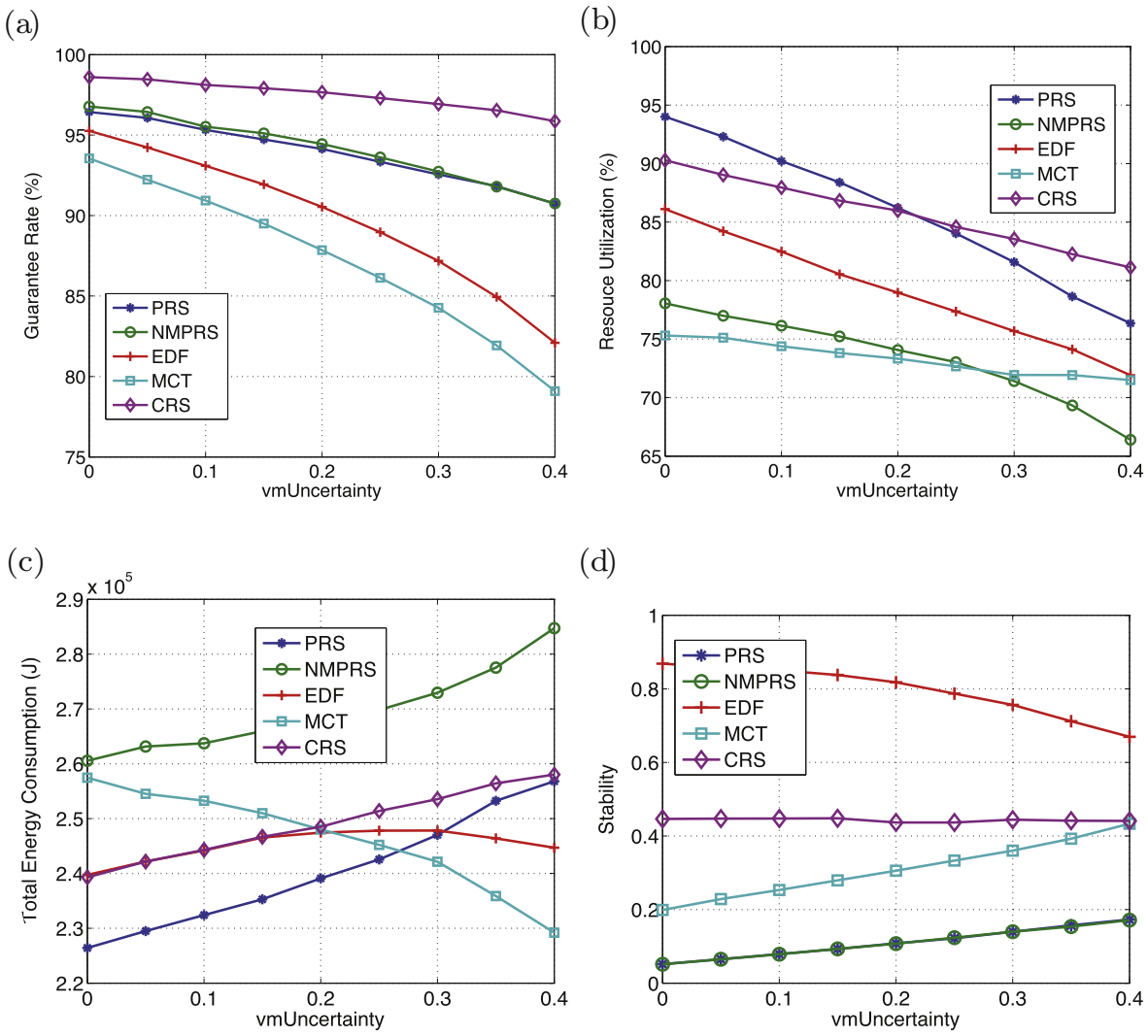


Fig. 7. Performance impact of vm Uncertainty.

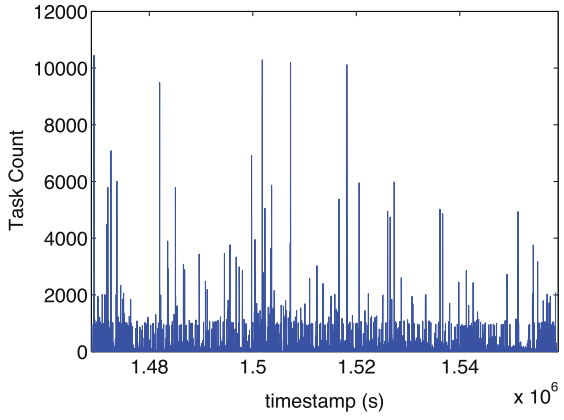


Fig. 8. The task distribution over time for Google traces.

- The deadline of each task is rounded up to 10% more of its maximum execution time. Other parameters are assigned in the way described in Section 5.1.

Table 4 shows the comparative results of the five algorithms based on the Google cloud workload traces.

Table 4
Performance comparison using Google cloud tasks.

Metric	Algorithm				
	PRS	NMPRS	EDF	MCT	CRS
Guarantee ratio	89.07%	90.68%	86.31%	85.83%	92.38%
Resource utilization	0.9059	0.4783	0.7839	0.5669	0.9381
Total energy($\times 10^8 J$)	2.2148	3.6160	2.7534	3.1369	2.3548
Stability	0.1043	0.1051	0.7223	0.1717	0.1659

From the above table, algorithm PRS, NMPRS, EDF, MCT and CRS have high guarantee ratios (89.07%, 9068%, 86.31%, 85.83% and 92.38%, respectively). The high guarantee ratios for all algorithms may be because of the supercomputing capacity of cloud data centers. However, there still exist some tasks that cannot be completed before their deadlines. This can be due to the following two reasons. First, the uncertainties of tasks' computing length and VMs' computing capacity make the execution time of the tasks longer, but their deadlines are fixed. Secondly, the time overheads of scaling up computing resources may lead to violation of some tasks' timing constraints. It is worthwhile noting that regarding to resource utilization, those algorithms (i.e., PRS, EDF and CRS) that utilize VM migration policies when scale down computing resources are as efficient on the Google data as on synthetic traces, but algorithms NMPRS and EDF perform worse on the Google data than on the

synthetic data. This can be contributed to the fact that the workload in real traces varies significantly as shown in Fig. 8, thus the computing resources required for the workload in the system also varies significantly. This result indicates that our strategies for scaling up and down computing resources can improve the resource utilization of cloud data centers in practice. In addition, PRS outperforms NMPRS, EDF and MCT by 89.40%, 15.56% and 59.80%, respectively, in terms of the resource utilization. The energy consumed by the system is more than that in the previous synthetic workload case. It is reasonable because the task count in Google workload traces is far larger than that in previous synthetic workloads, and the average task length of real-world tasks is much larger than that of synthetic tasks, which definitely results in more energy consumption. Furthermore, the stability of PRS, NMPRS, EDF, MCT and CRS are becoming small as compared to the synthetic workload case, which is especially evident for CRS and EDF. Obviously, the stability of CRS and EDF is less than that in synthetic workloads. We attribute it to the fact that the tasks' deadlines in Google traces are tight.

6. Conclusions and future work

In this paper, we investigate how to reduce the system's energy consumption while guaranteeing the real-time constraints for green cloud computing where uncertainty of task execution exists. We proposed an uncertainty-aware scheduling architecture for a cloud data center, and developed a novel scheduling algorithm, namely PRS, to make good trade-offs among tasks' guaranteeing ratio, system's resource utilization, system's energy consumption and stability. To evaluate the effectiveness of PRS, we conducted extensive simulation experiments with both the synthetic workloads and Google workload traces. Experimental results showed the effectiveness of the algorithm PRS compared with other related algorithms (NMPRS, EDF, MCT, CRS).

As a future research direction, we will aim at implementing and validating our strategies in a real-world cloud computing environment. Further, we plan to study a way of improving the precision of estimated task execution time. We expect that accurately estimating execution time leads to good scheduling decisions.

Acknowledgements

This research was supported by the National Natural Science Foundation of China under grant (no. 71271213 and 91024030) and Public Project of Southwest Inst. of Electron. & Telecom. Technology (2013001). In addition, we greatly appreciate all the anonymous reviewers for their constructive comments and suggestions.

References

Ahmad, F., Vijaykumar, T., 2010. Joint optimization of idle and cooling power in data centers while maintaining response time. ACM SIGPLAN Notices 45 (3), 243–256.

Anderson, D., Sweeney, D., Williams, T., Camm, J., Cochran, J., 2013. Statistics for Business & Economics. Cengage Learning.

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al., 2010. A view of cloud computing. Commun. ACM 53 (4), 50–58.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A., 2003. Xen and the art of virtualization. ACM SIGOPS Oper. Syst. Rev. 37 (5), 164–177.

Beloglazov, A., Abawajy, J., Buyya, R., 2012. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener. Comput. Syst. 28 (5), 755–768.

Beloglazov, A., Buyya, R., 2012. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr. Comput.: Pract. Exp. 24 (13), 1397–1420.

Beloglazov, A., Buyya, R., 2013. Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. IEEE Trans. Parallel Distrib. Syst. 24 (7), 1366–1379.

Berral, J.L., Gavalda, R., Torres, J.,2011. Adaptive scheduling on power-aware managed data-centers using machine learning. In: IEEE/ACM 12th International Conference on Grid Computing. IEEE, pp. 66–73.

Bruneo, D., 2014. A stochastic model to investigate data center performance and QoS in IAAS cloud computing systems. IEEE Trans. Parallel Distrib. Syst. 25 (3), 560–569.

Burge, J., Ranganathan, P., Wiener, J.L.,2007. Cost-aware scheduling for heterogeneous enterprise machines (CASHEM). In: IEEE International Conference on Cluster Computing. IEEE, pp. 481–487.

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R., 2011. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw.: Pract. Exp. 41 (1), 23–50.

Cameron, K., Ge, R., Feng, X., 2005. High-performance, power-aware distributed computing for scientific applications. Computer 38 (11), 40–47.

Garg, S.K., Yeo, C.S., Anandasivam, A., Buyya, R., 2011. Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers. J. Parallel Distrib. Comput. 71 (6), 732–749.

Hagimont, D., Kamga, C.M., Broto, L., Tchana, A., De Palma, N.,2013. DVFS aware CPU credit enforcement in a virtualized system. In: Middleware 2013. Springer, pp. 123–142.

Hermenier, F., Lorca, X., Menaud, J.-M., Muller, G., Lawall, J.,2009. Entropy: a consolidation manager for clusters. In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. ACM, pp. 41–50.

Herroelen, W., Leus, R., 2005. Project scheduling under uncertainty: Survey and research potentials. Eur. J. of Oper. Res. 165 (2), 289–306.

Hsu, C.-H., Slagter, K.D., Chen, S.-C., Chung, Y.-C., 2014. Optimizing energy consumption with task consolidation in clouds. Inform. Sci. 258, 452–462.

Hu, M., Veeravalli, B., 2013. Requirement-aware strategies for scheduling real-time divisible loads on clusters. J. Parallel Distrib. Comput. 73 (8), 1083–1091.

Kong, X., Lin, C., Jiang, Y., Yan, W., Chu, X., 2011. Efficient dynamic task scheduling in virtualized data centers with fuzzy prediction. J. Netw. Comput. Appl. 34 (4), 1068–1077.

Koomey, J., 2011. Growth in data center electricity use 2005 to 2010. The New York Times 49 (3).

Li, D., Wu, J.,2012. Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms. In: IEEE 41st International Conference on Parallel Processing (ICPP). IEEE, pp. 430–439.

Li, J., Ming, Z., Qiu, M., Quan, G., Qin, X., Chen, T., 2011. Resource allocation robustness in multi-core embedded systems with inaccurate information. J. Syst. Archit. 57 (9), 840–849.

Luo, J., Rao, L., Liu, X.,2012. eco-idc: Trade delay for energy cost with service delay guarantee for internet data centers. In: Cluster Computing (CLUSTER), 2012 IEEE International Conference on. IEEE, pp. 45–53.

Ma, Y., Gong, B., Sugihara, R., Gupta, R., 2012. Energy-efficient deadline scheduling for heterogeneous systems. J. Parallel Distrib. Comput. 72 (12), 1725–1740.

Mell, P., Grance, T., 2009. The nist definition of cloud computing. Natl. Inst. Stand. Technol. 53 (6), 50.

Mills, A.F., Anderson, J.H.,2010. A stochastic framework for multiprocessor soft real-time scheduling. In: IEEE 16th International Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, pp. 311–320.

Moreno, I.S., Garraghan, P., Townend, P., Xu, J.,2013. An approach for characterizing workloads in google cloud to derive realistic resource utilization models. In: IEEE 7th International Symposium on Service Oriented System Engineering (SOSE). IEEE, pp. 49–60.

Oh, S.-H., Yang, S.-M.,1998. A modified least-laxity-first scheduling algorithm for real-time tasks. In: IEEE 5th International Conference on Real-Time Computing Systems and Applications. IEEE, pp. 31–36.

Pettey, C., 2007. Gartner Estimates ICT Industry Accounts for 2 Percent of Global CO_2 Emissions, Available from: https://www.gartner.com/newsroom/id/503867 (14, 2013).

Qiu, M., Sha, E.H.-M., 2009. Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. ACM Trans. Des. Autom. Electron. Syst. (TODAES) 14 (2), 25.

Rizvandi, N.B., Taheri, J., Zomaya, A.Y., 2011. Some observations on optimal frequency selection in DVFS-based energy consumption minimization. J. Parallel Distrib. Comput. 71 (8), 1154–1164.

Sengupta, A., Pal, T.K., 2009. Fuzzy Preference Ordering of Interval Numbers in Decision Problems, vol. 238. Springer.

Srikantaiah, S., Kansal, A., Zhao, F.,2008. Energy aware consolidation for cloud computing. In: Proceedings of the 2008 Conference on Power Aware Computing and Systems, vol. 10. USENIX Association.

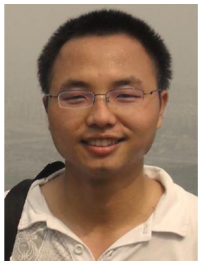
Van de Vonder, S., Demeulemeester, E., Herroelen, W., 2007. A classification of predictive-reactive project scheduling procedures. J. Sched. 10 (3), 195–207.

Van de Vonder, S., Demeulemeester, E., Herroelen, W., 2008. Proactive heuristic procedures for robust project scheduling: an experimental analysis. Eur. J. of Oper. Res. 189 (3), 723–733.

Van den Bossche, R., Vanmechelen, K., Broeckhove, J.,2010. Cost-optimal scheduling in hybrid IAAS clouds for deadline constrained workloads. In: IEEE 3rd International Conference on Cloud Computing (CLOUD). IEEE, pp. 228–235.

Xian, C., Lu, Y.-H., Li, Z., 2008. Dynamic voltage scaling for multitasking real-time systems with uncertain execution time. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 27 (8), 1467–1478.

Younge, A.J., Von Laszewski, G., Wang, L., Lopez-Alarcon, S., Carithers, W.,2010. Efficient resource management for cloud computing environments. In: IEEE International Conference on Green Computing. IEEE, pp. 357–364.
Zhu, X., Ge, R., Sun, J., He, C., 2013. 3e: energy-efficient elastic scheduling for independent tasks in heterogeneous computing systems. J. Syst. Softw. 86 (2), 302–314.



Huangke Chen received the B.S. degree in information systems from National University of Defense Technology, China, in 2012. Currently, he is a M.S. student in the College of Information System and Management at National University of Defense Technology. His research interests include task and resources scheduling in cloud computing and green computing.



Xiaomin Zhu received the B.S. and M.S. degrees in computer science from Liaoning Technical University, Liaoning, China, in 2001 and 2004, respectively, and Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2009. In the same year, he won the Shanghai Excellent Graduate. He is currently an associate professor in the College of Information Systems and Management at 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 TC, IEEE TPDS, JPDC, JSS and so on. He is also a frequent reviewer for international research journals, e.g., IEEE TC, IEEE TNSM, IEEE TSP, JPDC, etc. He is a member of the IEEE, the IEEE Communication Society, and the ACM.



Hui Guo received her BE and ME degrees in Electrical and Electronic Engineering from Anhui University and Ph.D. in Electrical and Computer Engineering from the University of Queensland. Prior to start her academic career, she had worked in a number of companies/organizations for information systems design and enhancement. She is now a lecturer in the School of Computer Science and Engineering at the University of New South Wales, Australia. Her research interests include Application Specific Processor Design, Low Power System Design, and Embedded System Optimization.



Jianghan Zhu was born in 1972. He received his M.S. and Ph.D. degrees in management science and technology from National University of Defense Technology, China, in 2000 and 2004, respectively. He is currently a professor and doctoral supervisor in College of Information Systems and Management in National University of Defense Technology. His research interests include combinatorial optimization, space-based information systems, and satellite application information link.



Xiao Qin received the B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology in 1992 and 1999, respectively. He received the PhD degree in computer science from the University of Nebraska-Lincoln in 2004. He is currently an associate professor in the Department of Computer Science and Software Engineering at Auburn University. Prior to joining Auburn University in 2007, he had been an assistant professor with New Mexico Institute of Mining and Technology (New Mexico Tech) for three years. He won an NSF CAREER award in 2009. His research is supported by the US National Science Foundation (NSF), Auburn University, and Intel Corporation. He has been on the program committees of various international conferences, including IEEE Cluster, IEEE MSST, IEEE IPCCC, and ICPP. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. He is a member of the ACM and a senior member of the IEEE.



Jianhong Wu is an internationally recognized mathematical authority in delay differential equations, infinite dimensional dynamical systems, non-linear analysis, mathematical biology and neural dynamics. His award-winning contributions continue to play a key role in advancing scientific research and development, and in assisting industry through “real-world” applications. He has published four major books and over 140 research papers in peer-reviewed journals, which are highly valued among scientific and industrial communities. He conducts comprehensive, multidisciplinary research in industrial and applied mathematics. He and his expert team are interacting and collaborating with university researchers in related fields, and with industrial data mining experts. Together, they are analyzing the information processing capabilities of neural networks modeled by differential equations to develop effective mathematical formulas and software for pattern recognition, classification and prediction. The team will also work with industrial partners to assess the effectiveness of their research discoveries in solving actual data analysis tasks.