# An Energy-Efficient Task Scheduling Heuristic Algorithm Without Virtual Machine Migration in Real-Time Cloud Environments

Yi Zhang[1(✉)], Liuhua Chen[2], Haiying Shen[2], and Xiaohui Cheng[1]

[1] School of Information and Engineering, Guilin University of Technology, Guangxi 541004, China
zywait@glut.edu.cn

[2] School of Electrical and Computer Engineering, Clemson University, Clemson 29631, SC, USA
liuhuachen@clemson.edu

**Abstract.** Reducing energy consumption has become an important task in cloud datacenters. Many existing scheduling approaches in cloud datacenters try to consolidate virtual machines (VMs) to the minimum number of physical machines (PMs) and hence minimize the energy consumption. VM live migration technique is used to dynamically consolidate VMs to as few PMs as possible; however, it introduces high migration overhead. Furthermore, the cost factor is usually not taken into account by existing approaches, which will lead to high payment cost for cloud users. In this paper, we aim to achieve energy reduction for cloud providers and payment saving for cloud users, and at the same time, without introducing VM migration overhead and without compromising deadline guarantees for user tasks. Motivated by the fact that some of the tasks have relatively loose deadlines, we can further reduce energy consumption by proactively postponing the tasks without waking up new PMs. In this paper, we propose a heuristic task scheduling algorithm called Energy and Deadline Aware with Non-Migration Scheduling (EDA-NMS) algorithm. EDA-NMS exploits the looseness of task deadlines and tries to postpone the execution of the tasks that have loose deadlines in order to avoid waking up new PMs. When determining the VM instant types, EDA-NMS selects the instant types that are just sufficient to guarantee task deadline to reduce user payment cost. The results of extensive experiments show that our algorithm performs better than other existing algorithms on achieving energy efficiency without introducing VM migration overhead and without compromising deadline guarantees.

**Keywords:** Virtualized cloud · Real-time task · Scheduling · Criticality · Energy-aware

## 1 Introduction

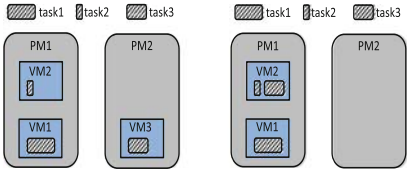
Cloud computing is one of the fastest evolving paradigm in the domain of computer science. Cloud serves as powerful computing platforms for a wide range


J. Chen et al. (Eds.): NSS 2016, LNCS 9955, pp. 80–97, 2016.
DOI: 10.1007/978-3-319-46298-1_6

of applications, such as meteorological prediction, genomic analysis, real-time complex physics simulations, monitoring watershed parameters through software services, and biological and environmental assistance [15]. Consequently, tens of thousands of hosts in a cloud datacenter consume enormous amount of energy. Therefore, reducing energy consumption has become an important task when deploying and operating cloud datacenters. In a virtual cloud computing environment, a set of submitted tasks from different users are scheduled on a set of virtual machines (VMs), and the task scheduling has become a critical issue for achieving energy efficiency. Previous energy-aware scheduling approaches [6,12,16,20,22] try to consolidate VMs to the minimum number of physical machines (PMs) to minimize the energy consumption, which however introduces high migration overhead. Figure 1(a) illustrates how existing task scheduling algorithms use VM migrations to further save energy. Suppose we are scheduling three tasks to the VMs in a datacenter. The first-in-first-out (FIFO) scheduling algorithm will create a VM for each task consequently. As each PM has two VM slots, the scheduling will end up with using two PMs as show in the figure. As task 2 is short, it will finish soon. After that, two tasks (e.g., task 1 and task 3) occupy two PMs. In order to reduce energy consumption, VM 3 can be migrated to PM 1 so that PM 2 can be shut down. A primary fraction of computing applications in cloud datacenters are real-time tasks [6], which have timing requirements on the response results. The arrival times of these tasks are dynamic and the predictions of their execution duration can also be difficult and sometimes impossible [3]. Users usually prefer that their task execution must be finished within a given deadline constraint. Motivated by the fact that some of the tasks have relatively loose deadlines, we can further reduce energy consumption by proactively postponing the tasks without waking up new PMs. Also, we no longer need VM migration to reduce energy consumption, thus the VM migration overhead is reduced. Figure 1(b) illustrates how arranging task with respect to their deadlines can help in eliminating VM migrations. Take the same scheduling problem as an example. As task 2 is short, we expect that it will finish executing soon. On the other hand, as task 3 does not have a stringent deadline, we can proactively postpone its execution. We schedule task 3 to VM 2 in PM 1. In this case, we no longer need the VM migration in Fig. 1(a) when VM 2 finishes execution.

In this paper, we propose a heuristic task scheduling algorithm called Energy and Deadline Aware with Non-Migration Scheduling (EDA-NMS) algorithm. EDA-NMS aims to provide a solution for achieving energy reduction for cloud providers without compromising deadline guarantees for user tasks. EDA-NMS exploits the looseness of task deadlines and tries to postpone the execution of the tasks with loose deadlines in order to avoid waking up new PMs.

In order to maximally satisfy user requests with different priorities, the proposed approach also introduces the concept of real-time criticality to accelerate the scheduling of priority tasks with stringent deadline constraints. Criticality is a different dimension than hard or soft characterization of a task, which is a measure of the cost of a failure(the higher the cost of failure, the more critical the




(a) Scheduling leads to VM migration. (b) Scheduling avoids VM migration.

**Fig. 1.** Task scheduling examples.

task) [13]. If two tasks have the same deadline, the task with higher criticality should be scheduled first. When determining the VM instant types, EDA-NMS selects the instant types that are just sufficient to guarantee task deadline to reduce user payment cost. EDA-NMS gives higher priority to guaranteeing task deadlines than reducing energy consumption.

The key contributions of this paper are as follows:

- We propose an energy-saving EDA-NMS algorithm that reduces the number of running PMs and avoids VM migration by exploiting the looseness of task deadlines without compromising task schedulability (i.e., the condition of being schedulable) and throughput.
- We conduct extensive simulation-based experiments to evaluate and analyze the performance of the proposed task scheduling algorithm. The results show that the proposed heuristic task scheduling algorithm not only reduces energy consumption, but also improves the completion time of real-time tasks.

## 2 Related Work

A significant amount of research efforts has been devoted to investigating the task scheduling in the cloud systems over last decade. Qiu *et al.* [16] studied the problem of assigning computing units to each task in a system to achieve energy savings at a minimum cost. Hosseinimotlagh *et al.* [12] proposed a VM scheduling algorithm based on the unsurpassed utilization level, which achieves optimal energy consumption while meeting a given QoS. It focuses on increasing the acceptance rate of arrival tasks but ignores the type of workloads running in VMs which affects the QoS guarantee of scheduling algorithm. Besides these works, most existing cloud schedulers, such as FIFO scheduling in Hadoop MapReduce [9], Fair scheduler in Facebook [7] and Capacity scheduler in Yahoo [21] schedule tasks based on worst-case execution time while ignoring dynamically changing cloud computing environments. As a result, they fail to fully utilize the resource. In other word, those approaches assume that cloud computing environments are deterministic and pre-computed schedule decisions will be statically followed during schedule execution. Unlike those approaches,

we leverage interval numbers to capture the dynamically changing cloud resource parameters to improve resource utilization.

Several scheduling works also address the problem of ensuring user deadlines as defined in Service Level Agreements (SLAs). Chen *et al.* [6] proposed a real-time scheduling strategy that allows executing only one task at any time instant on each VM. When the number of tasks increases, it needs vast VMs instants that will produce a lot of static energy consumption. Zhu *et al.* [22] presented a rolling-horizon optimization policy, which reduces energy consumption in virtualized data centers by supporting VM migration and VM placement optimization. These works reduce static energy consumption by migrating VMs between PMs. However, these works ignore the incurring VM migration overhead on the servers as well as the network infrastructure of the cloud. In contrast to previous researches, we propose the heuristic task scheduling algorithm to reduce static energy consumption. The total energy consumption consists of two parts: dynamic energy consumption and static energy consumption. The static energy consumption is the energy consumed by a host during idle time. The dynamic energy consumption is the extra energy consumed by a host when it is busy. As static energy consumption is dominant, our work focuses on reducing static energy consumption (i.e., reducing the number of active PMs). The proposed heuristic achieve reduce energy by selecting different types of VM instances with varying computing capacities for the tasks (i.e., we are actually adjusting execution speeds of real-time tasks, and consolidating these tasks into fewer number VMs, hence fewer number of PMs). As a result, it does not need to migrate VMs from an under-loaded host (PM) to other hosts (PMs). Furthermore, it also provides guarantees for the real-time tasks deadlines.

There are also some works that focus on the energy consumption model. The DVFS-enabled scheduling algorithms offers the minimum amounts of required CPU utilization to each task, and hence reduces the dynamic energy consumption as much as possible [5,18]. He *et al.* [10] developed a new energy-efficient adaptive scheduling algorithm (EASA) that can adaptively adjust supply voltages according to the system workload for dynamic energy efficiency. Most of previous research works like DVFS-enabled scheduling, focus on reducing the dynamic energy consumption as low as possible. However, the static power will last for a long time even for executing a low-speed task [11].

## 3 Model and System Architecture

In this section, we introduce the scheduling system architecture and three mathematical models: (i) the finishing time of a task, (ii) the laxity of a task, and (iii) the energy consumption of local a provider. These models will be used in our scheduling algorithm. Specifically, the scheduling system architecture is a VM based system, where VMs are launched for processing the submitted tasks and torn down when the tasks are finished. The system also dynamically turns on/off physical machines, and maintains the CPU utilization of the physical machines at the optimal level based on the number of VMs to reduce energy consumption.

(i) The finishing time of a task is calculated based on the length/size of the task and the computing capability of the VM to which the task is going to be assigned. It is used to estimate whether the task can be scheduled to a certain VM under deadline constrain.
(ii) The laxity of a task is calculated based on the task finishing time and its deadline. It is a measure of how urgent the task is. It is used for sorting the tasks in the scheduling queue.
(iii) The energy consumption is calculated based on the CPU utilizations of physical machines. Based on the estimated energy consumption, our algorithm dynamically turns on/off physical machines to reduce energy consumption.

# 4 Model and System Architecture

## 4.1 System Architecture

Figure 2 illustrates the compositional scheduling architecture used for the virtual cloud system.

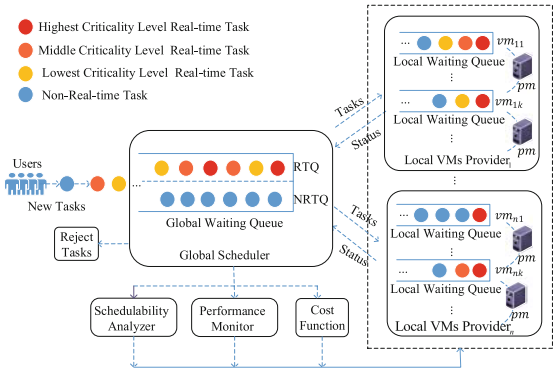



**Fig. 2.** Compositional scheduling architecture.

The scheduling architecture consists of two critical parts (the global scheduler and local VM providers) and three sub-components (performance monitor, schedulability analyzer and cost function), as shown in Fig. 2.

– The performance monitor observes the current workload in the system, checks the system status information such as currently allocated VMs, collects actual tasks processing time and arrival pattern information.
– The schedulability analyzer maintains and updates the configuration parameters which record tasks' deadlines and arrival time information provided by performance monitor. Also, the schedulability analyzer takes tasks from users and generates VMs startup plan from configuration parameters for different users.

– The cost function sub-component calculates the computing expense of executing tasks in the public cloud using price model offered by Amazon's Elastic Compute Cloud (EC2), based on the size of tasks and the computing price for renting a VM resource [3].

Cloud service providers (CSPs), who own large datacenters and server clusters, are incentivized by profits that they accrue by charging the end users for the service access [8]. CSPs provide services to end users through local VM providers. Each local VM provider is responsible for allocating cloud resource to the tasks of an individual user. One local VM provider offers one user massive computing power, needed storage and different services based on an SLA. Several neighboring local VM providers may form the CSPs with network connections. CSPs consist of a set of local VM providers: $LP = \{lp_1, lp_2, ..., lp_n\}$.

Each $lp_j$ manages a set of PMs: $PM^j = \{pm_1^j, pm_2^j, ..., pm_k^j\}, k = 0, 1, ..., |PM^j|$. A PM can host one or more VMs. For each local VM provider $lp_j$, it manages a set of PMs which contains a set of VMs: $VM^j = \{VM_1^j, VM_2^j, ..., VM_k^j\}, k = 1, ..., |PM^j|$. $VM_k^j$ is a set of VMs in $pm_k^j$ that belongs to $lp_j$, and $VM_k^j = \{vm_{k1}^j, vm_{k2}^j, ..., vm_{kr}^j\}, r = 1, ..., |VM_k^j|$. $vm_{kr}^j$ is the $r$th VM on PM $pm_k^j$ that belongs to $lp_j$. The resource demands of a set $VM^j$ cannot exceed the resource capacity of $PM^j$ which belongs to $lp_j$.

To satisfy tasks diversity, multiple task queues are employed in global scheduler. Thus, the real-time tasks are assigned to global real-time waiting queues (RTQ), and non-real-time tasks are assigned to global non-real-time queue (NRTQ). In RTQ, all the real-time tasks are sorted by their laxity values in an ascending order, whenever a new real-time task arrives. The task with the smallest laxity which means a level of urgency is first considered for execution in scheduling. The motivation of sorting the tasks in RTQ based on their laxities is that tasks with tight deadlines are processed earlier than others in order to avoid SLA violations.

**Definition 1.** *The laxity $\zeta_i$ of real-time task $\tau_{ikr}^j$ belong to $vm_{kr}^j$ means a level of urgency, and it is given as [6]:*

$$\zeta_{ikr}^j = d_i^u - (et_{ikr}^j)^+ - t_c, \tag{1}$$

*where $d_i^u$ is the deadline of task $\tau_i^u$, $(et_{ikr}^j)^+$ represents the maximal execution time of task $\tau_i^u$ executing on $vm_{kr}^j$ and $t_c$ is the current time.*

If some tasks have the same laxity value, then these tasks are sorted by their criticalities again. The tasks with higher criticality should be scheduled first. In NRTQ, the non-real-time tasks are sorted by their arrival times. The task with an earlier arrival time is scheduled first. Only when the RTQ is empty, the global scheduler schedules the tasks in NRTQ.

The local VM provider is bound to a specific user, so its VM instances can promote user context preservation, security and privacy. In other words, any VM instance of a local VM provider is dedicated to a single user until the instance is shut down when it approaches multiples of full hour operation (i.e., keep the

instance busy doing work until the charging time interval is due) and no tasks are running on it. For example, since the instance is charged based on the unit of hour, we shut down the VM only when it is idle and reaching multiples of full hour operation. One local VM provider $lp_j^u$ manages and monitors all pending and ready VM instances belong to one user $u$.

### 4.2 Task Model and Characteristics

The information provided by the task is the input to our scheduling algorithm. We introduce the task model and its characteristics in this section.

The tasks are submitted by individual users. We denote the set of tasks belonging to a separate user $u$ as $T^u = \{\tau_1^u, \tau_2^u, ..., \tau_m^u\}$. The tasks considered in this paper can be divided into two types: real-time and non-real-time trivially parallel tasks which are independent and aperiodic. Each task requires to be executed in one VM instance type and cannot be partitioned to multiple computing nodes. Each task $\tau_i^u$ is characterized by a 4-tuple of parameters: $\tau_i^u = (at_i^u, \widetilde{l}_i^u, d_i^u, k_i^u)$, where

- $at_i^u$ is the arrival time for task $\tau_i^u$.
- $\widetilde{l}_i^u$ is the length/size of task $t_i^u$, which is the number of instructions (i.e., millions instructions, MI). Note that the length of a task is uncertain before scheduling, but its lower bound $(l_i^u)^-$ and upper bound $(l_i^u)^+$ can be gained [2,6]. As in [6,17], we regard $\widetilde{l}_i^u$ as an interval number.
- $d_i^u$ is the deadline of $\tau_i^u$. Note that $d_i^u \geq at_i^u$. In this paper, the deadlines serve as the performance requirements specified by the users.
- $k_i^u \in \{K_1, K_2, K_3\}$ denotes the criticality of the task $\tau_i^u$.
  The set of criticality is a designation of the level of assurance against failure needed for a system component [4]. In this paper, we use three generic criticality levels. $K_1$ is the lowest criticality level, $K_3$ is the highest criticality level. The task with higher criticality level indicates that it is more important and usually requires urgent response.

### 4.3 VM Instances

VMs are categorized into $G$ distinct instance types: $it_1, it_2, ..., it_G$, and a VM of instance type $it_g$ denotes as $vm^g$. VM is configured to have a number of slots for executing tasks. For a given VM $vm_{kr}^j$, it is characterized by its $\widetilde{cap}_{vm_{kr}^j}$ and $\tau_{ikr}^j$, where $\widetilde{cap}_{vm_{kr}^j}$ denotes the CPU capacity represented by the number of instructions per second (MIPS), and $\tau_{ikr}^j$ is an indicator denoting that the task $\tau_i^u$ belong to $vm_{kr}^j$, respectively. Each user only specifies the types of VMs that are needed, but not the quantity of each requested VM type [8].

VM instance acquisition requests can be made at any time, but it may take startup time denoted as $\sigma$ for newly requested pending instance to be ready to use. Based on the previous research [14], $\sigma$ could take around 600 s from the arrival time of an instance acquisition request to the time when the VM is ready

to use. The overhead is paid by the user although a task cannot be executed on the VM during the time when the VM is starting up. Because cloud VM instances are currently billed by instance hours (rather than the exact user consumption time), the scheduling and scaling (i.e., whether wake up extra PMs) decisions should avoid partial instance-hour waste. Therefore, a reasonable policy is that whenever an instance is started, it is better to be shut down when the VM usage approaches full hour operation (i.e., keep the instance busy doing work until the charging time interval is due) [14].

As mentioned before, clouds now normally offer various instance types for users to choose, instead of offering one suit-for-all instance type. Correspond to Amazon EC2 instance type with the only exception that all the VMs are single-core, our experiments model multiple VM instance types of a cloud with different performance and varying costs as shown in following Table 1. The $it_1$ and $it_2$ are compute and memory optimized instance types which are most suitable for CPU and memory intensive applications respectively, like image processing, database systems and memory caching applications. The $it_3$ is a general instance type which provides the balance between compute and memory. General type instances are suitable for all general purpose applications. A computing intensive task can run faster on high-CPU VM instance than on high-memory VM instance. Choosing cost-effective instance types can both guarantee task deadlines and save user payment cost [14].

### 4.4 Task Finishing Time Estimation

The CPU capacity allocated to a task in a VM is measured in MIPS (million instructions per second) $\widetilde{cap}_{vm_{kr}^j}$, which arbitrarily varies over time. We do not know its actual value, but its lower and upper bounds can be obtained before scheduling [6]. As a result, the real execution time $\widetilde{et}_{ikr}^j$ cannot be exactly determined before scheduling. We utilize the interval number described in [6,17] to determine these uncertain parameters as follows.

$$\widetilde{cap}_{vm_{kr}^j} = [cap^-_{vm_{kr}^j}, cap^+_{vm_{kr}^j}], \tag{2}$$

**Table 1.** Characteristics of types of VMs used

| Type name | Description | Max MIPS | Cost | Startup lag |
|---|---|---|---|---|
| $it_1$ | High-CPU | 2500 | \$0.68/hour | $\sigma = 720$ s |
| $it_2$ | High-Memory | 2000 | \$0.50/hour | $\sigma = 720$ s |
| $it_3$ | General type | 1000 | \$0.085/hour | $\sigma = 600$ s |

where $cap^{-}_{vm^j_{kr}}$ and $cap^{+}_{vm^j_{kr}}$ are the computing capacity lower and upper bound of the VMs with minimal and maximal CPU performance.

$$\widetilde{et}^j_{ikr} = \frac{\widetilde{l}^u_i}{\widetilde{cap}_{vm^j_{kr}}} = \left[ \frac{(l^u_i)^-}{cap^+_{vm^j_{kr}}}, \frac{(l^u_i)^+}{cap^-_{vm^j_{kr}}} \right]. \tag{3}$$

$$\widetilde{ft}^j_{ikr} = \widetilde{st}^j_{ikr} + \widetilde{et}^j_{ikr}, \tag{4}$$

where $\widetilde{st}^j_{ikr}$ is the estimated start time of task $\tau^j_{ikr}$, and $\widetilde{ft}^j_{ikr}$ denotes the finish time of task $\tau^j_{ikr}$.

$$\widetilde{st}^j_{ikr} = max\{(ft^j_{ikr})^b, at^u_i\}, \tag{5}$$

where $(ft^j_{ikr})^b$ is the finish time of previously allocated task before $\tau^u_i$ executing on $vm^j_{kr}$.

### 4.5 Energy Consumption

**CPU resource utilization.** We first discuss the CPU resource utilization of the VMs and PMs, which are related to the energy consumption. The CSPs may offer different types of VM instances, which are suitable for different types of workloads. $lp_j$ is associated with an integer array $Q^j$ of $G$ members: $q^j_1, q^j_2, ..., q^j_G$, where $q^x_g$ indicates that number of type $g$ VMs ($vm^g$) are hosted on the PMs set $PM^j$ that belongs to $lp_j$. Since $Q^j$ is dynamic, it may change over time due to VM terminations and reconfigurations. We denote it as $Q^j(t)$ at time $t$. $lp_j$ contains a finite amount of computing resources $C^j_{cpu}$ coming from $PM^j$. The CPU resource utilization of $lp_j$ at time $t$ is denoted as $U^j(t)$, and the CPU resource requirement of $vm^g$ is denoted as $R^g_{cpu}$.

$$U^j(t) = \frac{\sum_{g=1}^{G} Q^j_g(t) \cdot R^g_{cpu}}{C^j_{cpu}} \times 100\,\% \tag{6}$$

$$= \frac{\sum_{k=1}^{|PM^j|} U^j_k(t)}{|PM^j|}, \tag{7}$$

where $U^j_k(t)$ is the CPU resource utilization of one PMs $pm^j_k$ belongs to $lp_j$ at time $t$.

The energy consumption (E) at a datacenter is defined as a total amount of power (P) consumed over a period of time (T) while performing the work [19].

$$E = P \cdot T. \tag{8}$$

The total energy consumption of $pm^j_k$ is denoted as $TE^j_k$. In CMOS chips, the total energy-consuming contains two main parts, one is the static energy

consuming $SE_k^j$ and the other is dynamic energy consuming $DE_k^j$. The $SE_k^j$ is the energy consumed during the idle time of $pm_k^j$. From [12], we can define $SE_k^j$ as:

**Static energy consumption.** The $SE_k^j$ is the energy consumed during the idle time of $pm_k^j$. From [12], we can define $SE_k^j$ as:

$$SE_k^j = \begin{cases} \frac{\gamma \cdot ME_k^j}{0,^{U_k^j}}, & when\ U_k^j > 0, \\ & otherwise. \end{cases} \tag{9}$$

where $ME_k^j$ is the energy consumed when a PM works with its maximum utilization, $\gamma$ is a constant ratio of the static energy consumption $SE_k^j$ to the maximum energy consumption $ME_k^j$ of $pm_k^j$ $(0 < \gamma < 1)$.

$$ME_k^j = P_{k\,max}^j \cdot t_{max}, \tag{10}$$

where $P_{k\,max}^j$ is the power consumed when a PM works with its maximum utilization, and $t_{max}$ is the time in which a PM works at its maximum computing power to finish a certain amount of tasks.

This ratio $\gamma$ depends on the physical characteristics of the PM, and it is constant during the time that a host is switched on.

**Dynamic energy consumption.** The relationship between dynamic energy consumption $DE_k^j$ and $U_j(t)$ is much more complex. From [11], we can know that several models proposed for the dynamic energy consumption $DE_k^j$ in the literature which are functions of the utilization of a PM. The dynamic energy consumption $DE_k^j$ and the total energy consumption $TE_k^j$ of $pm_k^j$ to execute tasks can be defined as:

$$DE_k^j = (ME_k^j - SE_k^j) \cdot U_k^j. \tag{11}$$

$$TE_k^j = SE_k^j + DE_k^j \tag{12}$$

$$= [\gamma + (1-\gamma) \cdot {U_k^j}^2] \frac{P_{k\,max}^j \cdot t_{max}}{U_k^j}. \tag{13}$$

Equation (13) explains that CPU resource utilization $U_k^j$ is the only adjustable parameter that has an impact on total energy consumption $TE_k^j$.

In the following, we will show that the static energy consumption dominates the total energy consumption. In Fig. 3, we assume the $\gamma = \{0, 0.2, 0.4, 0.8\}$, it means that the static energy consumption $SE_k^j$ takes 0 %, 20 %, 40 %, 80 % of the maximum energy $ME_k^j$. We compare the total energy consumption $TE_k^j$ under different static energy consumption by ignoring the exact value of $P_{k\,max}^j$. Figure 3 shows that the $pm_k^j$ consumes a noticeable amount of static energy for a long-life computing. Therefore, the static energy plays a profound role in the total energy consumption, and sometimes it comprises up to 70 % of total energy consumption [11]. When the value of $\gamma$ is larger, the static energy consumption is

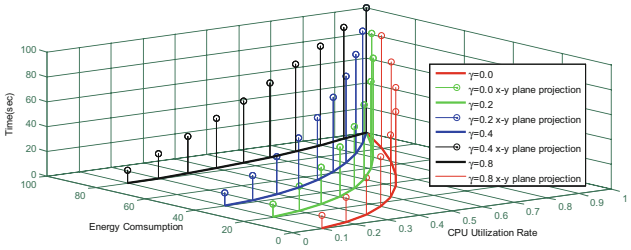



**Fig. 3.** Total energy consumption with various values of $\gamma$.

more than those with smaller $\gamma$. From the projection curves on x-y plane under various $\gamma$ values, we can see that the increasing speeds of energy consumption are different. The smaller $\gamma$ has faster increasing speed of energy consumption. This explains that most of total energy consumption is dynamic energy consumption when the $\gamma$ is smaller.

Only alleviating the dynamic energy consumption cannot reduce the total energy consumption significantly as long as there is high static energy consumption. Previous research works focus on consolidating VMs to alleviate the static energy consumption. However, the migration process imposes a high overhead depending on the network infrastructure. In addition, the source local VM provider spends more computing power during the live migration transient interval which might result in SLA violations. In order to handle this problem, we propose a task scheduling strategy that launches as few VMs (hence few PMs) as possible to guarantee most of tasks' deadlines and to enhance the CPU utilization level to minimize the static energy consumption.

## 5 Scheduling Strategy with Deadline Guarantee

Our energy-aware scheduling strategy focuses on how to finish all the submitted tasks before user specified deadlines with as few VM instance hours as possible.

**Definition 2.** *The computing cost of running a task $\tau_i^u$ on provider $lp_j$ with instance VM type $it_i \in \{it_1, it_2, it_3\}$ is defined as $C_{task}(\tau_i^u, lp_j, it_i)$.*

$$C_{task}(\tau_i^u, lp_j, it_i) = (\lceil \frac{\widetilde{l_i^u}}{\widetilde{cap}_{vm_{jk}}} \rceil \cdot C_{it_i}), \tag{14}$$

*where $\lceil \frac{\widetilde{l_i^u}}{\widetilde{cap}_{vm_{jk}}} \rceil$ means that the execution time is rounded up to the nearest discrete time unit (i.e. 1 h) of $lp_j$'s billing interval for cost calculations, $C_{it_i}$ denotes as the cost of running the VM instance $vm_{kr}^j$ of type $it_i$ on $lp_j$ for one time unit. When the deadline of task $\tau_i^u$ is not met, the computing cost is $\infty$.*

Our scheduling policy first schedules the task $\tau_i^u$ within deadline constraint on the local waiting queue of the cheapest (low hourly-cost) VM instance type among all the live instances. A task $\tau_i^u$ will be removed from the cheapest VM instance type queue to more expensive (high hourly-cost) VM instance type queue from the all available instance types, when the current state of $vm_{kr}^j$ is not able to finish task $\tau_i^u$ before its deadline. Even when there is no workload, a cloud application will always maintain at least one running VM instance. When one VM instance $vm_{jk}$ is approaching full hour operation, we need to decide whether to shut down the machine or not. The detail pseudocode of our EDA-NMS algorithm is showed in Algorithm 1.

**Algorithm 1.** Pseudocode of EDA-NMS algorithm

```
1: RTQ ← NULL;
2: NRTQ ← NULL;
3: for each new task τ_i^u do
4:   if d_i^u! = NULL then
5:     RTQ ← τ_i^u;
6:   else
7:     NRTQ ← τ_i^u;
8:   end if
9:   while RTQ! = NULL do
10:    sort all the tasks in RTQ by laxity in ascending order;
11:    if more than two tasks have the same ζ_i then
12:      sort all the tasks with same ζ_i by criticality k_i^u in descending order;
13:    end if
14:    τ_i^u ← get the task at the head in RTQ;
15:    the global scheduler assigns task τ_i^u to specific lp_j belonging to one user;
16:    move τ_i^u to the tail of local waiting queue on most cost-efficient vm_kr^j;
17:    execute LRTS algorithm (Algorithm 2);
18:  end while
19:  while RTQ == NULL && NRTQ! = NULL do
20:    τ_i^u ← get the task at the head in NRTQ;
21:    the global scheduler assigns task τ_i^u to specific lp_j belonging to one user;
22:    move τ_i^u to the tail of local waiting queue on most cost-efficient vm_kr^j;
23:    for each task in local waiting queue on vm_kr^j do
24:      execute τ_i^u;
25:    end for
26:  end while
27: end for
```

The pseudocode of local real-time scheduling (LRTS) algorithm is shown in Algorithm 2.

**Algorithm 2.** Pseudocode of LRTS algorithm

```
τ_i^u ← get the task at the head in local waiting queue on vm_kr^j;
calculate the start time st̃_ikr^j, execution time et̃_ikr^j and finish time ft̃_ikr^j of τ_i^u;
while ft̃_ijk^u > d_i^u do
    Find next cost-efficient vm_kr'^j;
end while
if can find one vm_kr^j on lp_j make ft̃_ijk^u ≤ d_i^u then
    while τ_i^u != NULL do
        execute τ_i^u;
        τ_i^u ← get the task at the head in local waiting queue on vm_kr^j;
    end while
else
    reject τ_i^u
end if
```

## 6 Performance Evaluation

### 6.1 Environment Setup

To demonstrate the performance improvements gained by our EDA-NMS algorithm, we quantitatively compare it with a existing algorithms PRS using the CloudSim simulator [6]. We compare the user payment cost and the completion time of real-time tasks of running cloud applications. In the simulation framework, we can also control the input parameters, such as workload patterns and task deadlines. The detailed parameters are given as follows:

- The simulation environment consists of a datacenter with 10000 hosts, where each host is modeled to have a single CPU core (with CPU performance 3000 MIPS, 3500 MIPS and 4500 MIPS), 4 GB of RAM memory, and 1 TB of storage [1,6].
- We employ the interval number to control a task's deadline, which can be calculated as:

$$d_i^u = at_i^u + \frac{U\left[(l_i^u)^-, (l_i^u)^+\right]}{cap^+_{vm_{kr}^j}} + U[0, 500]\, s, \tag{15}$$

  where $U[0, 500]$ s denotes a variable that subjects to uniformly distributed from 0 s to 500 s, and it determines whether the deadline of a task is loose or not.
- We randomly generated the task's length: $\widetilde{l}_i^u = [5000, 100000]$ MIs in a uniform distribution.
- The arrival of tasks follows Poisson distribution at the arrival rate of $Poisson(\lambda), \lambda = 4$ per unit of time, it means the arrival interval between two consecutive tasks obey the negative exponential distribution with parameter $Exp(1/\lambda)$.

### 6.2 Performance Under Changing Workloads

In these experiments, we focus on two types of workload. The single type workload experiment for compute-intensive tasks whose main bottleneck is CPU's computing power. In the mix type workload evaluation, we simulated three types of tasks, including mix, computing intensive and I/O intensive. The processing time parameters of single type and mix type workload experiment on different types VM instance are summarized in Table 2.

**Table 2.** Mix type workload unit execution time

| | Mix | Computing intensive | I/O intensive |
|---|---|---|---|
| General | 50 s/MI | 50 s/MI | 50 s/MI |
| High-CPU | 25 s/MI | 15 s/MI | 50 s/MI |
| High-Memory | 25 s/MI | 50 s/MI | 15 s/MI |

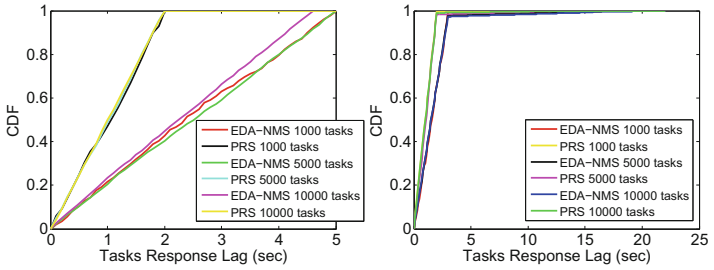
Three task trace groups are generated for the experiments, each includes 1000, 5000 and 10000 tasks and the number of real-time tasks, non-real-time tasks and reject tasks produced by the experiment results is summarized in Table 3.

**Table 3.** Some running results of single type workload/mix type workload

| | Number of real-time tasks | Number of non-real-time tasks | Reject number of real-time tasks |
|---|---|---|---|
| EDA-NMS(1000 tasks) | 584/587 | 416/413 | 0/0 |
| PRS(1000 tasks) | 587/581 | 413/419 | 0/0 |
| EDA-NMS(5000 tasks) | 2915/2912 | 2085/2088 | 0/0 |
| PRS(5000 tasks) | 2910/2913 | 2090/2087 | 1/0 |
| EDA-NMS(10000 tasks) | 5838/5835 | 4162/4165 | 5/2 |
| PRS(10000 tasks) | 5828/5832 | 4172/4168 | 6/2 |

To investigate the duration of the real-time tasks executions, we use the cumulative distribution function (CDF) of the response time lags (i.e., deadline - finish time) of the tasks. From the experiment result of Fig. 4, we can see the EDA-NMS and PRS scheduling algorithms' performance under these two types of workload conditions. EDA-NMS outperforms PRS in terms that it achieves bigger tasks response lag than PRS, it means that the completion time of the real-time tasks executed by EDA-NMS is shorter than PRS under different types of workloads.

By analysing the situation of the rejected tasks, we can know the system stability under EDA-NMS and PRS. The reject tasks of EDA-NMS consist of



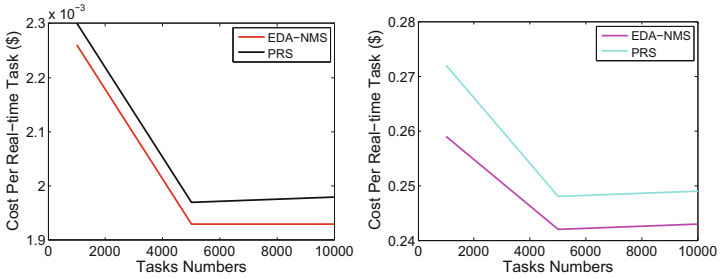

(a) Computing Intensive Tasks. (b) Mix Type Tasks.

**Fig. 4.** Tasks response time lag.

7 tasks with middle criticality, and the reject tasks of PRS consist of 8 tasks including 3 ones with highest criticality, 4 ones with middle criticality and 2 ones with lowest criticality. The task with different criticality which misses the deadline has different influence on the system stability. So we can find that there are fewer rejected tasks with higher criticality of our EDA-NMS than PRS. It means that our EDA-NMS has better system stability than PRS.

### 6.3 Cost Efficient Comparison

By changing the task number from 1000 to 10000, we first use the average execution cost of real-time tasks to compare the performance of these two scheduling policies (EDA-NMS, PRS). The experiment results in Fig. 5 show that our EDA-NMS has lower average execution cost in both two computing intensive and mix type workload cases.




(a) Computing Intensive Task. (b) Mix Type Task.

**Fig. 5.** Average cost per real-time task comparison.

We then compare the total cost of these two scheduling policies (EDA-NMS, PRS) with three different types of VM instances as shown in Table 1. From the comparison results of total cost are illustrated in Table 4, we can see that the

**Table 4.** Mix workload cost

| No | Number of tasks | VM types | Total cost($) | Static energy consumption |
|---|---|---|---|---|
| Choice 1 | 10000 | General, General | $9.95 \times 10^3$(58 % higher) | Standard |
| Choice 2 | 10000 | General, High-Memory | $7.74 \times 10^3$(23 % higher) | Standard |
| Choice 3 | 10000 | General, High-CPU | $7.4 \times 10^3$(17 % higher) | Standard |
| Optimal | 10000 | General, High-CPU, High-Memory | $6.3 \times 10^3$ | 1.5 standard |

choice 1, choice 2 and choice 3 incur 58 %, 23 % and 17 % more cost than the optimal solution separately. Our choice 3 is closest to the optimal cost, and it outperforms other two choices. Although the optimal solution can obtain the lowest cost, its statics energy consumption is 1.5 times of other three choices. Because the future workload cannot be known in advance, so the optimal cost can't be obtained in real life. Hence, our choice 3 is cost and energy efficient solution.

## 7 Conclusion

In this paper, EDA-NMS exploits the looseness of task deadlines and tries to postpone the execution of the tasks that have loose deadlines in order to avoid waking up new PMs. It incrementally provides cloud resources to the tasks based on the user specified deadlines, the estimated completion times of the tasks, and the resource utilization levels of the hosts. The results of extensive experiments show that our approach perform better than other existing approaches on achieving energy efficiency without introducing VM migration overhead and without compromising deadline guarantees. In the future, we are going to implement EDA-NMS in a real testbed and evaluate its performance.

**Acknowledgments.** The work on this paper has been supported by Scientific and Technological Research Program for Guangxi Educational Commission grants ♯2013YB113, Guangxi Universities key Laboratory Fund of Embedded Technology and Intelligent Information Processing.

## References


1. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency Comput. Pract. Experience **24**(13), 1397–1420 (2012)
2. Berral, J.L., Gavalda, R., Torres, J.: Adaptive scheduling on power-aware managed data-centers using machine learning. In: Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing. pp. 66–73. IEEE Computer Society (2011)
3. Van den Bossche, R., Vanmechelen, K., Broeckhove, J.: Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In: Proceedings of CLOUD, pp. 228–235. IEEE (2010)
4. Burns, A., Davis, R.: Mixed criticality systems-a review. Department of Computer Science, University of York, Technical report (2013)
5. Calheiros, R.N., Buyya, R.: Energy-efficient scheduling of urgent bag-of-tasks applications in clouds through DVFS. In: Proceedings of CloudCom, pp. 342–349. IEEE (2014)
6. Chen, H., Zhu, X., Guo, H., Zhu, J., Qin, X., Wu, J.: Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment. J. Syst. Softw. **99**, 20–35 (2015)
7. Facebook. https://www.facebook.com/
8. Gao, Y., Wang, Y., Gupta, S.K., Pedram, M.: An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems. In: Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, p. 31. IEEE Press (2013)
9. Hadoop MapReduce. https://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html
10. He, C., Zhu, X., Guo, H., Qiu, D., Jiang, J.: Rolling-horizon scheduling for energy constrained distributed real-time embedded systems. J. Syst. Softw. **85**(4), 780–794 (2012)
11. Hosseinimotlagh, S., Khunjush, F.: Migration-less energy-aware task scheduling policies in cloud environments. In: Proceedings of WAINA, pp. 391–397. IEEE (2014)
12. Hosseinimotlagh, S., Khunjush, F., Samadzadeh, R.: Seats: smart energy-aware task scheduling in real-time cloud computing. J. Supercomput. **71**(1), 45–66 (2015)
13. Mall, R.: Real-Time Systems: Theory and Practice. Pearson Education, India (2009)
14. Mao, M., Li, J., Humphrey, M.: Cloud auto-scaling with deadline and budget constraints. In: Proceedings of GRID, pp. 41–48. IEEE (2010)
15. Pop, F., Dobre, C., Cristea, V., Bessis, N., Xhafa, F., Barolli, L.: Deadline scheduling for aperiodic tasks in inter-cloud environments: a new approach to resource management. J. Supercomput. **71**(5), 1754–1765 (2015)
16. Qiu, M., Sha, E.H.M.: Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. ACM Trans. Des. Autom. Electron. Syst. (TODAES) **14**(2), 25 (2009)
17. Sengupta, A., Pal, T.K.: Fuzzy preference ordering of intervals. In: Sengupta, A., Pal, T.K. (eds.) Fuzzy Preference Ordering of Interval Numbers in Decision Problems. STUDFUZZ, vol. 238, pp. 59–89. Springer, Heidelberg (2009)
18. Tang, Z., Qi, L., Cheng, Z., Li, K., Khan, S.U., Li, K.: An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment. J. Grid Comput. 1–20 (2015)

19. Veni, T., Bhanu, S.: A survey on dynamic energy management at virtualization level in cloud data centers. Comput. Sci. Inf. Technol. **3**, 107–117 (2013)
20. Wang, W.J., Chang, Y.S., Lo, W.T., Lee, Y.K.: Adaptive scheduling for parallel tasks with QoS satisfaction for hybrid cloud environments. J. Supercomput. **66**(2), 783–811 (2013)
21. Yahoo. https://www.yahoo.com/
22. Zhu, X., Yang, L.T., Chen, H., Wang, J., Yin, S., Liu, X.: Real-time tasks oriented energy-aware scheduling in virtualized clouds. TOCC **2**(2), 168–180 (2014)