Journal of Artificial Intelligence and Data Mining (JAIDM), Vol. 9, No. 2, 2021, 213-226.

Shahrood University of Technology

Journal of Artificial Intelligence and Data Mining (JAIDM)
Journal homepage: http://jad.shahroodut.ac.ir

Research Paper

An Energy-aware Real-time Task Scheduling Approach in a Cloud Computing Environment

Hossein Momeni* and Nahid Mabhoot

Department of Computer Engineering, Golestan University, Gorgan, Iran.

Article Info

Article History:
Received 09 December 2020
Revised 15 February 2021
Accepted 06 March 2021

DOI: 10.22044/jadm.2021.10344.2171

Keywords:
Task, Real-time, Cloud Computing, Scale-up, Scale-down, Scheduling.

**Corresponding author:h.momeni@gu.ac.ir (H. Momeni).*

Abstract

The interest in cloud computing has grown considerably over the recent years, primarily due to the scalable virtualized resources. Thus cloud computing has contributed to the advancement of the real-time applications such as the signal processing, environment surveillance, and weather forecast, where time and energy considerations are critical in order to perform the tasks. In the real-time applications, missing the deadlines for the tasks will cause catastrophic consequences Thus real-time task scheduling in a cloud computing environment is an important and essential issue. Furthermore, energy-saving in the cloud data center, regarding the benefits such as the reduction in the system operating costs and environmental protection is an important concern that has been considered during the recent years, and is reducible with an appropriate task scheduling. In this paper, we present an energy-aware real-time task (EaRT) scheduling approach for the real-time applications. We employ the virtualization and consolidation techniques subject to minimizing the energy consumptions, improve resource utilization, and meeting the deadlines of the tasks. In the consolidation technique, the scale-up and scale-down of the virtualized resources could improve the performance of task execution. The proposed approach comprises four algorithms, namely energy-aware task scheduling in cloud computing (ETC), vertical VM scale-up (V2S), horizontal VM scale-up (HVS), and physical machine scale-down (PSD). We present the formal model of the proposed approach using timed automata in order to prove precisely the schedulability feature and correctness of EaRTs. We will show that our proposed approach is more efficient in terms of the deadline hit ratio, resource utilization, and energy consumption compared to the other energy-aware real-time tasks scheduling algorithms.

1. Introduction

A cloud computing environment is a distributed computing model based on the virtualization technology. It is also known as a dynamic service provider with the ability of scalability, flexibility, and virtualization of resources on the internet [1-3]. This environment contains the physical and virtual resources and virtualization platforms, and also allows the users and administrators to do the migration of a virtual machine (VM) from one server or physical machine (PM) to another PM. Migration of a VM from one host to another one helps to balance the workload among PMs running in the cloud data center.

In particular, the scalable virtualized resources in cloud computing address the high computation demands that are required by the real-time

systems. In other words, the employment of computing resources to meet the user's requirements makes cloud computing an attractive and suitable environment for executing the real-time tasks [4].

Thus a very important issue regarding the subject that many applications are deployed on clouds has a real-time nature. It means that the correctness of a real-time application is not only dependent on the computation results but also on the time the results are available. For some latency-sensitive applications, providing a real-time guarantee is even a necessity. For example, in an environment surveillance, forecasting or medical simulation has a deadline constraint, and when it fails, other results may be useless [5-7].

In these applications, the deadline meet has a priority over the other criteria so presenting the task schedulability model on the clouds is important. As we know, most of the previous real-time task scheduling algorithms reject the tasks if they are not guaranteed by using the available VMs and increase the number of virtual machines (horizontal scaling) on PMs in a deadline constraint.

On the other hand, regarding the rapid growth of cloud computing, development of using cloud computing services, and interest of the customers in these services, the providers create a data center on a large scale including thousands of VMs as the computing nodes, which result in a huge consumption of energy with high expenses, and here, consuming energy is an important and vital concern [6, 11-13].

When a large number of real-time tasks arrive at short intervals to the cloud system, there is a need to set up PMs and deploy more virtual machines in order to respond to these demands. However, the overhead time of deploying new physical and virtual machines that have a long delay in starting the time of real-time tasks may cause the violation of the deadline of some real-time tasks.

Moreover, as the system workload increases, the number of virtual machines will increase, and as the system workload decreases, the virtual machines can merge and consolidate into fewer PMs. Turning off the idle virtual and PM is important in terms of the energy management problems, especially in the green cloud computing. The consolidation techniques aim to consolidate the tasks in fewer PMs and VMs. The consolidation schemes should optimize the resource utilization in such a way as to avoid violating the service level agreements (SLA), energy consumption, and performance degradation [4].

In this paper, we propose an energy-aware task scheduling approach, namely EaRTs, for the real-time tasks, which is independent and dynamic. In the EaRTs approach, we present an energy-aware real-time task scheduling (ETC) algorithm. We then present the horizontal VM scale-up (HVS) algorithm in order to guarantee that the deadline of the tasks is met in an active VM. The horizontal VM scale-up is done by increasing the number of VMs through creating a new VM and deploying it on a PM.

The vertical scaling-up of VM increases the number of resources including the processor, memory, and storage of a VM, and can be done in less than a few milliseconds, while the horizontal scaling-up can take several minutes. Thus we propose the vertical VM scale-up (V2S) algorithm, and finally, to reduce the number of active PMs in the system to save energy, we present the physical machine scale-down (PSD) algorithm using the consolidation technique. In our proposed algorithms, the virtualization technique is used to decrease the energy consumption in the cloud data center. Besides, from the elasticity viewpoint, VM's vertical scaling is used when there is no possibility to guarantee the tasks by increasing horizontal scaling in their deadline constraint. Furthermore, in the proposed approach, by executing more tasks over active PMs, the resource utilization rate is increased. The EaRTs approach creates a balance among three factors including the guarantee rate, resource utilization, and decreasing energy consumption. We also present three models, namely the resource model, task model, and energy model, in order to describe this proposed approach and formalize our approach using timed automata [8] to verify all the proposed algorithms and employ a temporal logic based on a timed automata to the description of schedulability [9, 10]. Timed automata is a dominant method for modeling and verification of the real-time systems. Using timed automata, we model and verify our approach, and check the schedulability, deadlock-freeness, and correctness of our proposed algorithms.

The rest of this paper is organized as what follows. Section 2 provides a brief background and the definitions. Section 3 presents some notable related works on the real-time task scheduling. Section 4 presents our system models. Section 5 presents our proposed approach. Section 6 presents the formal models of the proposed approach. Section 7 presents the verification of our proposed approach. Finally, Section 8 concludes the paper.

2. Definitions

2.1. Scalability

Scalability is an important feature in the cloud computing environments such that the cloud services and computing platforms can be scalable regarding the geographical locations, hardware performance, and software configuration [14].
In general, there are two kinds of resource scalability technologies in cloud computing. The first one is horizontal scaling, which determines the number of VMs, and the second one is vertical scaling, which is determined by changing the partition of resources (CPU, memory, storage, etc.) inside VMs. Vertical scaling can be done in less than a few milliseconds, and VM cannot be used at once [15].

2.2. Schedulability

Schedulability checks that all tasks have reached their deadline and exist in the right conditions. In other words, it checks whether all paths are in the conditions in which no task is placed in the error state. If the answer is yes, it guarantees that the system is schedulable in all conditions [17]. A system with a set of tasks with constraint resources are called schedulable if no execution satisfying the constraints of the system violates a deadline [16].

2.3. Timed Automata

Timed automata is a tuple *(L, L0, LF, Σ, C, E, I)*, where:

L is a finite set of locations;
L_0 is a subset of *L* and a set of initial locations;
L_F is a subset *L* and a set of final locations;
Σ is a set of finite alphabets;
C is a finite set of clocks with real negative values;
E: $E \in L \times \Phi(C) \times (\Sigma \cup \{\epsilon\}) \times 2C \times L$ is a set of edges;
I: $L \rightarrow \Phi(C)$ is a mapping of local invariants to locations.

Timed automata is a state transition system adding clocks with real values [18]. The states are called locations, which model different possible configurations of the system. The transition among these locations shows the way that the system can progress. The locations can be labeled with some features, which show the elapsed time for transition from a previous location to a new one.
An example of timed automata is shown in Figure 1 with *x, y* clocks. Whenever a guard exists, the system will be transferred from the location *l0* to the location *l1*, and *x* clock will become zero.

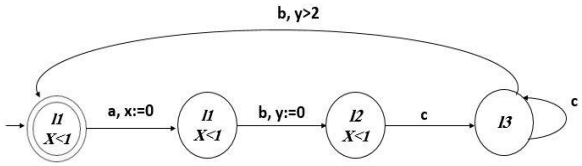

Figure 1. A timed automaton with 2 clocks.

$X < 1$ in the *l1, l2* locations guarantee that the system with guard *c* can be transferred from the location *l2* to the location *l3*, when about one time unit has passed from the event *a*. When guard *b* exists, the system will be transferred from the location *l1* to the location *l2*, and *y* clock will become zero. When guard *d* exists and 2 time units have passed of clock *y*, the system will be transferred from the location *l3* to the location *l0*, and guarantees that the delay between *b* and d is more than two time units.

3. Related Works

In the recent years, the issue of high energy consumption in a cloud environment has received much attention, and therefore, the energy-aware scheduling algorithms have been developed. Nevertheless, a few of them support the real-time task scheduling with the management of energy, and give a guarantee for the schedulability of the system.
Chen *et al*. [5] have presented ERES as a scheduling algorithm aware of energy for the real-time tasks in the cloud environment based on the EDF policy. They used the VM consolidation technique in order to decrease PMs and save energy. Zhu *et al*. [6] have presented an algorithm, namely EARH, in order to balance between the consumed energy and the schedulability tasks. In this algorithm, virtualization is used in order to decrease the consumed energy. Chen *et al*. [19] have proposed the PRS algorithm, where each VM just places one task in its local queue, and whenever the number of arriving tasks increases, more VMs are required to guarantee the real-time tasks. This algorithm uses the virtualization method in order to decrease the consumed energy. This algorithm only considers one task execution at a time on a VP. As the number of tasks increases, more VMs are required, which increases the static energy consumption. In [11], the ERECT algorithm has been proposed for the real-time task scheduling in virtual cloud regarding saving energy. At first, this algorithm guarantees the real-time tasks with the least active PMs, and whenever the workload

of the system increases and there is the need to turn on a new PM, the PM with less energy consumption would be selected to decrease the consumed energy. If the workload of the system decreases, it uses the consolidation of VMs and turn off the idle PMs in order to decrease the consumed energy. In this algorithm, the results obtained are just based on the simulation, and there is no verification on guaranteeing the task execution within their deadlines.

Jun *et al*. [20] have presented the EART algorithm for the energy-aware task scheduling. This algorithm while guarantees the limitation of the deadline, also considers saving energy. The tasks in the waiting queue are placed and sorted according to the urgency level, which decreases the rate of deadline miss. The indivisible task goes to VMs with the least consumption of energy, and the divisible tasks go to several VMs in order to improve scheduling and save energy. In EART, the task is not accepted if it is not guaranteed using the existing VMs and increasing the number of VMs (horizontal scalability) within its deadline.

Zhang *et al*. [21] have presented the EAD-NMS algorithm regarding saving energy for scheduling the real-time tasks, which decreases the number of PMs and prevents migration of VMs with tasks for which their deadline is not very sensitive. In this algorithm, in order to postpone the execution of the task that has a soft deadline causes some new tasks, with a hard deadline to enter, and leads to reject these tasks, and as a result, the tasks guarantee that the rate is reduced.

Hosseini *et al*. [22] have presented the SEATS algorithm for scheduling VMs in order to maximize the level of utilization by presenting more CPU performance for the host VMs. In this algorithm, if using the current processor is less than the optimal threshold, the VM scheduler shares the remaining of a million instruction per second (MIPS) among VMs to maximize MIPS in order to reach the optimal utilization level. This algorithm is only applicable when the system load is low, and if the number of tasks increases, there will be a slight improvement in the energy consumption when completing the tasks with a hard deadline.

Wang *et al*. [23] have proposed the FESTAL algorithm, which is a real-time task scheduling algorithm in the cloud and improves fault tolerance and resource utilization in the cloud. The main purpose of FESTAL is to increase the schedulability of the system, and to use the resource until it reaches the fault tolerance. This algorithm considers the task schedulability regardless of the energy criterion.

Given this background, in most of the previous research works, the real-time task scheduling algorithm will not accept the task if it is not guaranteed by using the existing VMs and increasing the number of VMs (horizontal scaling) in its deadline. We will present an energy-aware real-time task scheduling approach based on the consolidation technique subject to minimizing the energy consumptions, improving the resource utilization, and meeting the deadlines of the tasks.

4. System Models

In this section, we will describe our proposed models formally in order to present our approach.

4.1. Resource Model

There is a set of physical machines (PM = {pm_j, j = 1, 2, ..., n} n = |PM|) in the cloud environment with a high scale data center. Also on each pm_j, there is a set of virtual machines (VM_j = {vm_{j1}, vm_{j2}, ..., vm_{jk}}, k = |VM_j|). We model each pm_j as (1):

$$pm_j = \{M_j, R_j, E_j, VM_j\} \quad (1)$$

where M_j is the processor speed in MIPS, R_j is the capacity of memory, E_j is the maximum energy of PM, and VM_j is a set of VMs in pm_j.

We model each virtual machine vm_{jk} as (2):

$$vm_{jk} = \{M_{jk}, R_{jk}, P_{jk}\} \quad (2)$$

where M_{jk} is the processor speed in MIPS, R_{jk} is the capacity of memory assigned to vm_{jk}, and P_{jk} is the readiness time of vm_{jk}.

Equation (3) shows that the total amount of resources required for VMs on a PM should not be more than the capacity of the PM resources.

$$M_j \geq \sum_{k=1}^{|vm_j|} M_{jk} \;\&\&\; R_j \geq \sum_{k=1}^{|vm_j|} R_{jk} \forall pm_j \in PM \quad (3)$$

where M_j is the total processor speed of PM_j and R_j is the total memory capacity of PM_j.

4.2. Task Model

We consider a finite set of real-time and independent tasks (T = {t_1, t_2, …, t_m}). Each task is defined as a four partite tuple in (4):

$$t_i = < a_i, lh_i, d_i, f_i > \quad (4)$$

where a_i is the task arrival time, lh_i is the task length (in million instruction), d_i is the deadline, and f_i is the task finish time of t_i.

We calculate the execution time of t_i on v_{jk} as (5):

$$et_{ijk} = lh_i / M(v_{jk}) \quad (5)$$

where lh_i is the length of *ti* and $M(v_{jk})$ is the processor speed that assigns to v_{jk}.
The finish time of t_i on v_{jk} is calculated as (6):

$$ft_{ijk} = et_{ijk} + bt_{ijk} \quad (6)$$

where bt_{ijk} is the beginning time of the task execution and et_{ijk} is the length of the task execution of t_i on v_{jk}.
The beginning time of the task execution bt_{ijk} is computed as (7):

$$bt_{ijk} = \max\{a_i, p_{jk}\} \quad (7)$$

where a_i is the task arrival task and p_{jk} is the readiness time of v_{jk}.
At first, creating and deploying of VM on a PM is determined by the beginning time of its task, and for each t_i that assign to v_{jk}, it will be updated as (8):

$$p_{jk} = bt_{ijk} + et_{ijk} \quad (8)$$

In this work, our main goal is to maximize the rate of the real-time task acceptance that has a priority in comparison to any other criteria in such tasks. It can be defined by (9):

$$Max \frac{TotalAcceptTask}{|T|} \quad (9)$$

where *TotalAcceptTask* is the total number of the executed tasks that meet their deadlines, and is defined as (10):

$$TotalAcceptTask = \sum_{i=1}^{|T|} \sum_{j=1}^{|PM|} \sum_{k=1}^{|vm_j|} t_{ijk} \quad (10)$$

If task t_i is assigned to v_{jk} and is terminated without a missing deadline, then t_{ijk} equals to 1; otherwise, it will be 0.

4.3. Energy Model

We define the energy consumption model as (11):

$$\text{TotalEnergyConsumption} = \sum_{j=1}^{|PM|} \int_{bt}^{ft} (k.E_j.ac_j^t + (1-k).E_j.u(t))dt. \quad (11)$$

where *bt* and *ft* are the beginning time and finishing time of task execution, respectively, and *k* is a fraction of energy consumption rate and we consider it 70% [24], E_j is the maximum energy of PM, $ac^t_{j \in \{0,1\}}$ if pm_j is active in time t, and *u(t)* is the utilization of pm_j in time t.
In this work, the next goal is to minimize the energy consumption of task execution that finishes before the deadline. It can be defined by (12):

$$Min(\frac{TotalEnergyConsumption}{TotalAcceptTask}) \quad (12)$$

In order to minimize energy consumption, we employ the VM consolidation technique and turn off the idle PMs [3], which will be described in Section 5.
The next goal in this work is to maximize the resource utilization. It can be defined by (13).

$$\frac{MAX\ (TotalAcceptTaskLength)}{(TotalActiveHostMips \cdot TotalActiveTime)} \quad (13)$$

that is computed from the executed task total length over the total processor performance of active PMs in the system during the execution. *TotalAcceptTaskLenghth* is the executed task total length, and is calculated as (14), and *TotalActiveHostMips* is the total processor performance of active PMs, and is calculated as (15).

$$TotalAcceptTaskLength = (\sum_{j=1}^{|PM|} \sum_{k=1}^{|V_j|} \sum_{i=1}^{|T|} lh_i.t_{ijk}) \quad (14)$$

where lh_i is the length of t_i, and if t_i assigns to v_{jk}, then t_{ijk} will be equal to 1; otherwise, it will be equal to zero.

$$TotalActiveHostMips = (\sum_{j=1}^{|PM|} M_j.wt_j) \quad (15)$$

where wt_j is the total time of pm_j when it is activated during the experiment, and M_j is the speed of the pm_j 's processor.

5. Energy-aware Real-time Task Scheduling Approach

In this section, we describe our energy-aware task scheduling approach for real-time tasks (EaRTs) in a cloud computing environment.. Firstly, we present the ETC algorithm and then present the HVS, V2S and PSD algorithms.

5.1. ETC Algorithm

When a new task arrives at the system, it is placed in a general queue initially. At first, this algorithm is checked according to the number of existing VMs in the system, whether the task can be finished without a missing deadline.
If it finds several VMs in order to execute the task without missing the deadline, the task will be assigned to a VM with a less turnaround time. If execution of the task in its deadline is guaranteed by active VMs in the system, the horizontal VM scale-up should be done (if execution of the task can be guaranteed on a VM).
If by the horizontal VM scale-up the task cannot finish before its deadline, the vertical VM scale-up should be done, and then the task will be assigned to a VM, which guarantees its execution before the deadline; otherwise, the task will not be accepted. Figure 2 shows the pseudo-code of the ETC algorithm.

```
ETC algorithm
Successtage = false;
Select_VM = Null;
For each new task ti, do
    Add ti to the general queue
End for
Sort (tasks, in general, queue by increasing the deadline);
For each ti in general queue, do
    For each vjk available do
        Calculate btijk and execution time etijk of task ti;
        If btijk + etijk = < di, then
            Success_Tage = True;
        End if
    End for
    If Success_Tage = True, then
        Select_VM = Select vjk with minimal finish time to execute ti ;
        Re-calculate the readiness time of vjk;
    End if
    If Success_Tage = False, then
        Select_VM = HVS();
    End if
    If Select_VM = Null, then
        Select_VM = V2S();
    End if
    If Select_VM! = Null, then
        Assign ti to Select_VM for execute;
        Re-calculate the readiness time of Select_VM;
        Success_Tage = True;
    End if
    If Success_Tage = False, then
        Reject ti;
    End if
End for
```

Figure 2. ETC algorithm.

5.2. Horizontal VM Scale-up (HVS Algorithm

We have designed HVS to guarantee that the deadline of the tasks is met in the active VM. Horizontal VM scale-up is done by increasing the number of VMs through creating a new VM and deploying it on a PM. The pseudo-code of the HVS algorithm is shown in Figure 3.

```
HVS algorithm
Vjk = Select (a type of virtual machine with Min_mips (VMs type) is the minimal processor performance of VMs Possible for executing ti within its deadline).
Success_Tag = False;
If Vjk! = Null, then
    PmList = Sort (active PM is available in the increasing order of the remaining processor capacity).
    For each pmj in PmList, do
        If Remaining_Mips (pmj) >= M(VjK), then
            Deploy vjk on pmj;
            Remaining_Mips (pmj) = Remaining_Mips (pmj)– M(vjk)
            Calculate the readiness_time (Vjk )
            Success_Tag = True; Break;
        End if
    End for
    If Success_Tag = False, then
    PmList = Sort (inactive PM is available in the increase order of the energy power)
    For each pmj in PmList, do
        If vjk can be guaranteed finishing ti within its deadline, then
            Turn on an inactive PM and deployt vjk on it.
            Remaining_Mips (pmj)=Remaining_Mips(pmj) – M(vjk)
            Calculate the readiness_time (Vjk )
            Success_Tag = true;
        End if
    End For
    End if
End if
```

Figure 3. HVS algorithm.

After an appropriate VM selection that can guarantee the execution of the real-time task t_i in its deadline, VM will be deployed on a PM with a maximum utilization. After deploying VM, its readiness time (rt_{jk}) is calculated as (16):

$$rt_{jk} = ct + dpt(v_{jk}) \quad (16)$$

where ct is the current time and $dpt(v_{jk})$ is the time of creation and deployment of v_{jk} on PM. If VM cannot be deployed in this way, an inactive PM with the least energy consumption in which VM can be deployed on it will be turned on, and VM will be created on it. In this situation, the readiness time of VM is calculated as (17):

$$rt_{jk} = ct + tt(pm_j) + dpt(v_{jk}) \quad (17)$$

where $tt(pm_j)$ is the turning on time of pm_j and $dpt(v_{jk})$ is the deployment time of v_{jk} on pm_j.

5.3. Vertical VM Scale-up Algorithm (V2S)

Actually, vertical VM scale-up increases the amount of resources including the processor, memory, and storage of a VM. Vertical scaling-up of VM can be done in less than a few milliseconds, while horizontal scaling-up can take time for several minutes.

In the HVS algorithm, creating more new VMs may miss the deadline of tasks so vertical scaling-up of VMs should be done, as shown in Figure 4.

V2S finds a VM with a possibly minimal processor capacity required to execute the task in its deadline constraint such that it can increase the processor capacity as follows.

If the remaining of the PM processor capacity can guarantee the execution of the real-time task, V2S adds it to the VM processor capacity, and then assigns the task to this VM; otherwise, V2S computes the extra amount of processor capacity allocated to other VMs deployed on this PM (in a manner that the allocated task deadline to these VMs will not be missed). If the remaining of the processor capacity of PM plus an extra amount of other VMs processor capacity equals the amount of the required processor capacity of the real-time task, V2S, at first, selects this capacity from the remaining capacity of PM, then it selects from VMs that are ordered according to increasing of waiting tasks on them. Finally, V2S assigns the task for execution to VM.

Among the other VMs deployed on PMs, V2S calculates the extra amount of each VM processor capacity, and then selects VMs in the order of increasing the waiting tasks and reduces their processor capacity in order to provide the amount of processor capacity required by the selected VM. Finally, V2S assigns the task for execution to VM.

```
V2S algorithm
1.  Find_Tag = False;
2.  For each v_jk available, do
3.      Calculate Extra Mips of v_jk that
        tasks on v_jk can be finished before
        its deadline and require Mips of execute t_i
        before its deadline on v_jk;
4.  End For
5.  VmList = Sort (VMs by Increase require Mips)
6.  For each v_jk in VmList, do
7.      If Find_Tag = False, then
8.          If (remaining_Mips of pm_j >=
                    Require_Mips of vm_jk), then
9.              MIPS( vm_jk) =Required Mips (vm_jk) + MIPS ( vm_jk);
10. ;
11.             Re-calculate the readiness time of vm_jk ;
12.             Select vm_jk to execute t_i
13.             Find_Tag = True; break;
14.             End if
15.         End if
16. End for
17.  If Find_Tag = False, then
18.     For Each v_jk in VmList, do
19.         If (total Extra Mips other vm on pm_j
                add remaining mips of pm_j >=
                    Required Mips of vm_jk) Then
20.                 MIPS (vm_jk) = Required Mips (vm_jk) + MIPS ( vm_jk) Re-calculate the readiness time of VMs;
21.                 Select vm_jk to execute t_i;
22.                 Find_Tag = True; break;
23.             End if
24.         End for
25. End if
26.  If Find_Tag = False, then
27.     For each v_jk in VmList, do
28.         If (total Extra Mips other vm on pm_j >=
                Required Mips of vm_jk) Then
29.             MIPS( vm_jk) = Required Mips (vm_jk) + MIPS (vm_jk);;
30.             Re-calculate the readiness time of VMs ;
31.             Select vm_jk to execute t_i;
32.                 Find_Tag = True; break;
33.             End if
34.         End for
```

Figure 4. V2S algorithm.

In the V2S algorithm, we calculate the readiness time of vjk whose processor capacity has changed as (18):

$$rt_{jk} = rt_{jk} - ert_{ijk(old\_mips)} + ert_{ijk(new\_mips)} - \sum_{i=1}^{|waitingtask\_list_{vjk}|} et_{ijk(old\_mips)} + \sum_{i=1}^{|waitingtask\_list_{vjk}|} et_{ijk(new\_mips)} \quad (18)$$

where rt_{jk} is the readiness time of v_{jk}, $ert_{ijk(old\_mips)}$ is the remaining execution time of t_i that is running on v_{jk} with the previous processor capacity, $ert_{ijk(new\_mips)}$ is the remaining execution time of t_i that is running on v_{jk} with a new processor capacity, $et_{ijk(old\_mips)}$ is the task execution time that is located in the local queue of v_{jk} and is processed by the previous processor capacity of VM, and finally, $et_{ijk(new\_mips)}$ is the task execution time that is located in the waiting local queue of v_{jk} and is processed by a new processor capacity of VM.

The processor capacity that v_{jk} requires to execute t_p in its deadline range is calculated as (19):

$$\mathrm{Re}qMips(v_{jk}) = IdealMips(v_{jk}) - M(v_{jk}) \quad (19)$$

where $M(v_{jk})$ is the processor capacity of v_{jk}, *IdealMips, and* (v_{jk}) is the processor capacity that if it is assigned to v_{jk}, it can execute all of its tasks plus t_p in their deadline constraints, and is calculated as (20):

$$IdealMips(v_{jk}) = \left(\sum_{i=1}^{|waitingtasklist_{vjk}|} lh(t_{ijk}) + ert(t_{zjk}) / (d(t_p) - ct) \right) \quad (20)$$

where ct is the current clock, $ert(t_{zjk})$ is the remaining execution time of running task t_{zjk} on v_{jk}, and $d(t_p)$ is the deadline of the new task t_p. If the maximum delay that all allocated tasks to v_{jk} can tolerate is more than zero, then the extra amount of processor capacity of v_{jk} is calculated as (21):

$$ExtraMips(v_{jk}) = M(V_{jk}) - NecessaryMips(v_{jk}) \quad (21)$$

where $NecessaryMips(v_{jk})$ is the processor capacity of v_{jk} that can execute all its assigned tasks without missing a deadline, and is calculated as (22):

$$NecessaryMips(v_{jk}) = \left(\sum_{i=1}^{|waitingtasklist_{vjk}|} lh(t_{ijk}) + ert(t_{zjk}) \right) / (rt(v_{jk}) - ct + Maxdelaytolerate(v_{jk})) \quad (22)$$

where $rt(v_{jk})$ is the readiness time of v_{jk} and ct is the current clock.

5.4. Physical Machine Scale-down Algorithm (PSD)

The PSD algorithm is used to reduce the number of active PMs in the system in order to save energy. The PSD algorithm uses VMs consolidation and removes the idle VMs to turn off the active PMs. PSD, at first, checks all of the existing VMs, and if more time has passed than the threshold of being idle of a VM, it removes that VM, and then PSD by using the VM migration technique, consolidates VMs in PMs as far as possible. Figure 5 shows the pseudo-code of PSD.

PSD algorithm

1. **For** each v_{jk} available **do**
2. **If** vm_{jk}'s idle time bigger than idle time maximum specified for VMs, **then**
3. Delete v_{jk};
4. **End if**
5. **End for**
6. Sort (active PM in an increasing order of the Remaining Processor Capacity).
7. **For** each active pm_j available **do**
8. If all the VMs deployment on pm can be migrated to other active PMs, **then**
9. Migrate all the VM deployment on pm_j to destination PMs;
10. **End if**
11. **End for**
12. **For** each active pm_j available **do**
13. **If** pm_j is idle, **then**
14. turn off pm_j;
15. **End for**

Figure 5. PSD algorithm.

To consolidate VMs, PSD arranges the active PMs by increasing the processor capacity remaining order, and begins by VMs of PMs with the least remaining of the processor capacity so that it can help them to migrate to other active PMs. If it finds a destination for all of the deployed VMs on a PM, it will migrate VMs to the destination PM. Finally, this algorithm will turn off all the idle PMs.

6. Formal Model of Proposed Approach

In this section, we present a formal model of our proposed approach in terms of timed automata and use a model checking approach in order to verify the soundness of our proposed approach formally. With model checking, a system is specified using a collection of timed automatons. Each automaton has a finite number of states and transitions between these states. The *Clocks* and *Boolean* expressions may guard these states and transitions [19]. *Chan* is used for synchronization in the system.

We use the UPPAAL 4.1.19 model checker, which can check different kinds of real-time system's features and is used in order to model and verify the properties of our proposed approach [25-27]. The UPPAAL query language uses the features that are a subset of CTL (computation tree logic) for checking [28]. The automatons of our model including the *Task Automaton*, *Scheduler Automaton*, and R*esource Automaton* are presented as follows.

6.1. Task Automaton

The task automaton shown in Figure 6 is the most important automaton in our model. Different states that the task passes from arrival to the system till existing are modeled by in this automaton. Since in the cloud computing environment, the users send their requests irregularly and the tasks have irregular arrival times, we assume that the tasks in this model are aperiodic and independent.

When a task arrives the system, it goes from the *Initial* state to the *Release* state, and it stays in the *Release* state. The *Clock* time is used for each task in order to define the time feature of the task. After passing the release time, the task will go from the *Release* state to the *Ready* state, and it

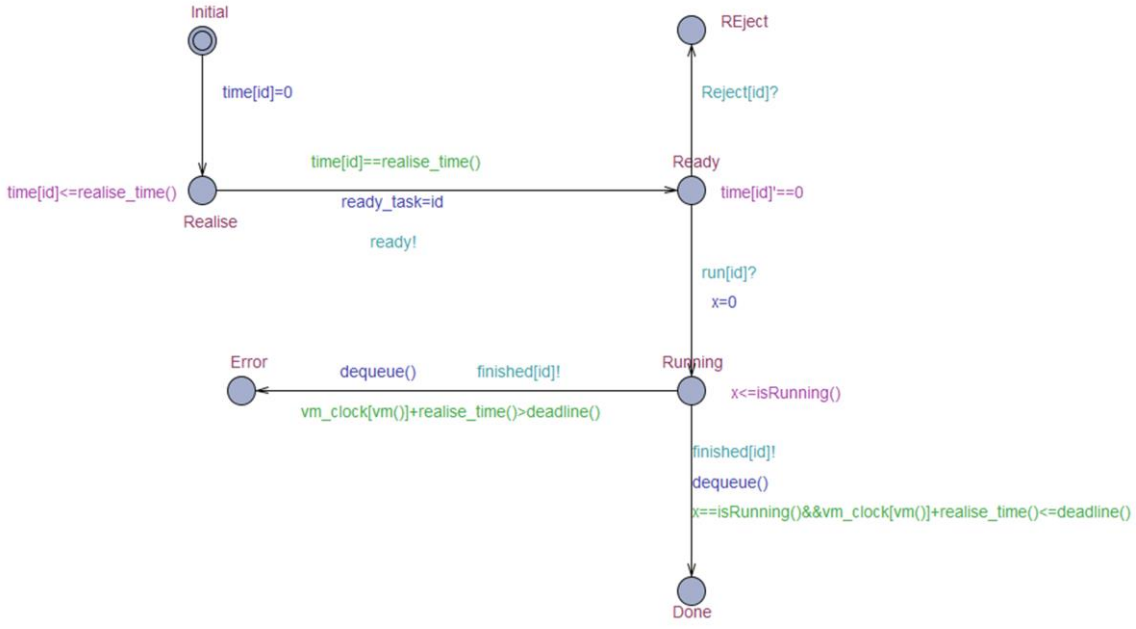

Figure 6. Task automaton model.

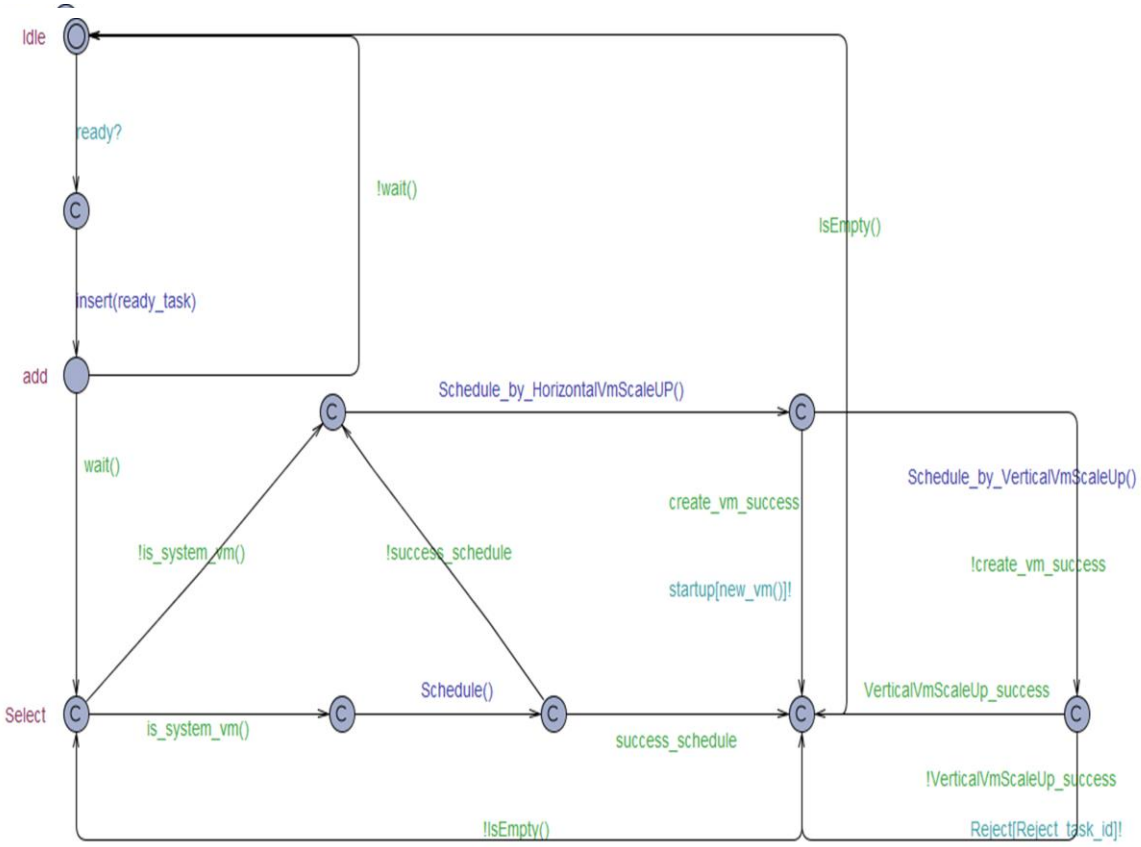

Figure 7. Scheduler automaton model.

informs the scheduler of its arrival by using the *ready* channel, and then goes to the general queue of the scheduler. The task stays in the *Ready* state till the scheduler assigns a resource to it and then it goes to the *Running* state. If the scheduler could not guarantee execution of this task in its deadline, it will go to the *Reject* state and will not be accepted by the system. The r*un* channel is used for synchronization between the task automaton and the resource automaton. The task will stay in the *Running* state till its execution on resource ends. The X clock is used for passing this time. After passing the time, if the task finishes in its deadline, it will go to the *Done* state, and if the deadline is missed, it will go to the *Error* state.

6.2. Scheduler Automaton

The *scheduler* automaton is depicted in Figure 7. The *Scheduler* locates the tasks that arrive the system in a general queue by their deadlines in an increasing order and then schedules them.

At first, the scheduler is in an *Idle* state and the general queue of the scheduler in which the tasks are located is empty. The *IsEmpty()* function is used to check the fullness or emptiness of the general queue. Whenever a new task arrives the system, the scheduler goes from the *Idle* state to the *Add* state. This automaton selects the task from the heading of the queue for scheduling and goes to the *Select* state. The scheduler by using the *is-system-vm()* function, checks if there is a VM in the system that the cloud executes the task in its deadline.

The *success_schedule* function checks the status of the allocated task execution on this VM. If this execution is not successful, the *create-vm-success* function checks whether by creating VM the deadline of the task will be guaranteed or not. If this creation meets the task deadline, the *startup* channel initiates a new VM creation and allocates task to this VM. Otherwise, the *VerticalVmScaleUp_Success* function checks whether by vertical scaling-up of VMs the task can be executed in its deadline, and then it will accept the task and guarantee its execution; otherwise, the task will not be accepted, and by using the *Reject* channel informs the task.

6.3. Resource Automaton

Figure 8 shows the automaton model of the resource. In this model, the resource can be used after passing its setting up time.

At first, the resource is in the *Initial* state, which means that it is not ready to use and it is not set

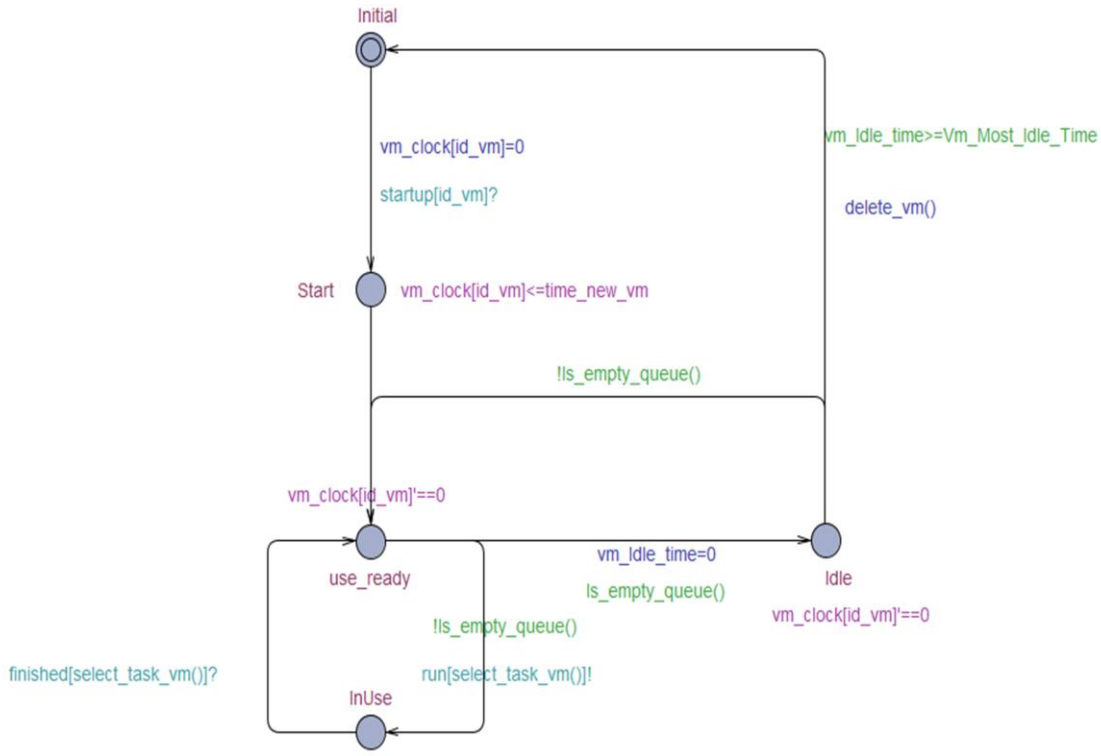

Figure 8. Resource automaton model.

up. The resource will go to the *Start* state by the *startup* channel to synchronize the resource and scheduler automaton. The *Vm-clock* is used for each resource in order to control its scheduling features and after passing this time, the resource is ready to use and goes to the *use-ready* state. In this state, the resource for each task that is in its queue goes to the *Inuse* state; it stays in this state for a duration of time that the task is running on it. The *Run* channel is used for synchronizing between the resource and the task automaton. When the task is selected of the general queue for execution, it will be informed by this channel, and when the time of execution of the task on resource finished, the *finished* channel is used for informing resource of finishing the task to go to the *use-ready* state. In this state, the *is-empty-queue()* function checks if the resource queue is empty, and it goes to the *Idle* state, and it stays in this state till a task is allocated to it, and then goes to the *use-ready* state, or the time is more than one threshold and decides to delete it and goes to the *Initial* state. The threshold is determined by *vm-most-idle-time* and *vm-idle-time* clock.

7. Verification Results

In this section, we present the analyzing results of our proposed approach, formally described in the previous section. The main goal of verification of the real-time task scheduling systems is the schedulability analysis via checking the safety and bounded liveness properties [29]. Schedulability expresses that a system, by the considering time constraints, scheduling policy, resources, and tasks would guarantee execution of all the accepted tasks in their deadline constraints.

The different formulas of the Timed Computation Tree Logic (TCTL) specification language that we have used in our models are as follows:

A [] φ: This formula denotes the safety property, and means that *for all paths φ holds on all states*.

A [] not (φ): This formula denotes the bounded liveness property, and means that there is no path and no state such that the system is in φ. Note that A [] not φ can represent deadlock freeness if φ is logically a deadlock property.

E<> Φ: This formula denotes that the *reachability* property is used for verification of schedulability. This property checks that a special event happens at least once in the system. Therefore, beginning from a primary state, it surely reaches the expected state. In fact, we use the reachability property for the schedulability analysis of the system in this work. If the *Error* state would not be reachable, the system will be schedulable.

According to these formulas, we check the deadlock-freeness, *reachability*, bounded liveness of the system modeled with timed automata

Reachability: We traverse the total space state in the proposed automatons using UPPAAL and

check whether all paths are in a condition that no task is located in the ERROR state or not. This property checks that a special event happens at least once in the system. Therefore, beginning from a primary state, it surely reaches the expected state. If the *Error* state would not be reachable, the system will be schedulable. We check that our proposed approach is schedulable with the following specifications in UPPAAL:

A [] forall (i: task_id) not TASK (i).Error

This formula checks that in all of the paths and states (total space state), considering time limitations, the location of ERROR cannot be reached from the task model; consequently, the system is schedulable.

E<> forall(i : task_id) TASK(i).Error

This formula checks that in all of the paths and states (total space state), considering time limitations, location of ERROR can be reached from the task model; consequently, the system is not schedulable.

Deadlock-freeness: One of the *safety* properties is the deadlock-freeness, which can be verified by the reachability analysis. Reachability checks that no possible state of a system ever reaches a deadlock and ensures that *something bad* never happens. We check that our proposed approach is deadlock-free with the following specification in UPPAAL:

Scheduler[] not deadlock

Task[] not deadlock

Resource[] not deadlock

Correctness: The main correctness criterion of our proposed approach is to ensure that whenever a task is submitted to a resource, the scheduler receives it eventually. Moreover, whenever the scheduler allocates a task to a VM, the VM receives it eventually.

We check the correctness of our proposed approach using the following specification in UPPAAL:

Task.Release --> Scheduler.Add

Resource.Start --> Task.Running

We have also evaluated the efficiency of our proposed ETC algorithm experimentally by comparing it with the ERECT [11] algorithm as a high related and popular work in terms of task deadline hit ratio and energy consumption of tasks and resource utilization. We also consider a non-migration version of ETC and ERECT, namely NM-ETC and NM-ERECT, and compare them with ETC.

We simulated our proposed approach using the *CloudSim* simulator [30]. We consider the simulation parameters as shown in Table 1.

We also assume:

- PM with processor performance, which is 2660 and 1860 MIPS and rate of energy consuming 250,400 w.
- Setting up time of a PM is 90 s and creation time of new VM on PM is 15 s.
- Length of tasks is in uniform between random [100000, 200000] million instructions.

Table 1. Simulation parameters.

Parameter	Fixed value	Variable value
Task count	1000	1000, 2000, 3000
Min deadline	100	50, 100, 150, 200
Task length	[100000-200000]	

Figure 9 shows the deadline hit ratio of ETC, ERECT, NM-ETC, and NM-ERECT. As it is shown in this figure, the deadline hit ratio in ETC has improved in comparison to ERECT, about 30% in average. Furthermore, NM-ETC has improved the deadline hit ratio in comparison to NM-ERECT about 34%. This improvement is because our EaRTs approach uses vertical scaling of VMs when horizontal scaling is not able to guarantee the real-time tasks.

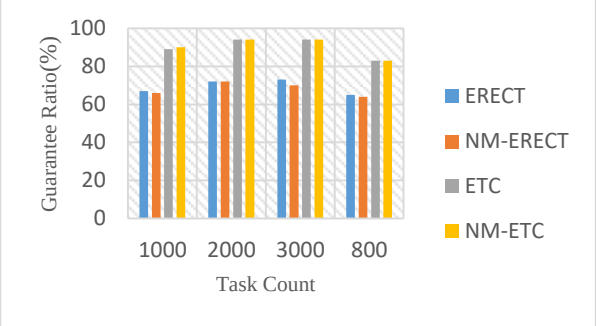

Figure 9. Deadline hit ratio comparison.

The ETC algorithm uses the processor speed of PMs (in Mips) as far as possible, and executes more real-time tasks on PMs; consequently, it has improved the resource utilization rate in comparison to ERECT, in average 18%. Also NM-ETC has improved the resource utilization rate in average 23% in comparison to NM-ERECT. These results are shown in Figure 10.

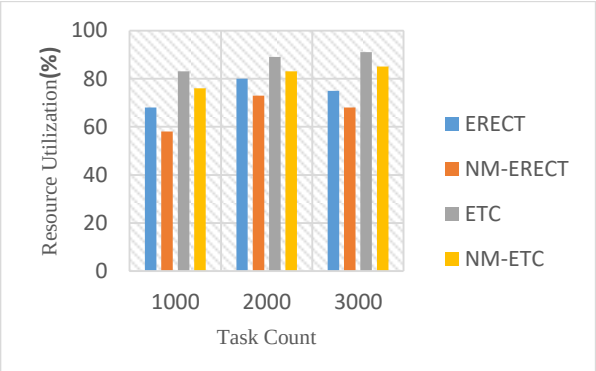

Figure 10. Resource utilization rate comparison.

As shown in Figure 11, the energy consumption of the accepted tasks in the ETC algorithm has improved 13.66% in comparison to ERECT. Moreover, NM-ETC in comparison to NM-ERECT has improved about 16.33% on average. Since we employed the VM consolidation technique and turned off the idle PMs in ETC, the energy consumption was decreased.

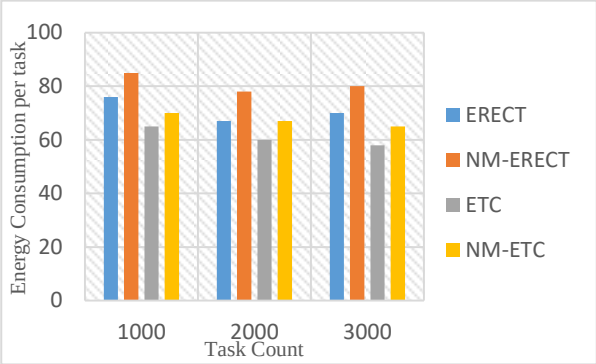

Figure 11. Energy consumption of accepted tasks.

We have also evaluated the impact of the real-time task deadline on the ETC algorithm in different lengths. As shown in Figure 12, in shorter deadlines, as ETC and NM-ETC algorithms, where using vertical scaling, have performed better than ERECT and NM-ERECT.

By increasing the deadline length, the ETC algorithm has improved the deadline hit ratio by an average 35.5% in comparison to ERECT. Moreover, NM-ETC has improved on average 37.25% in comparison to NM-ERECT.

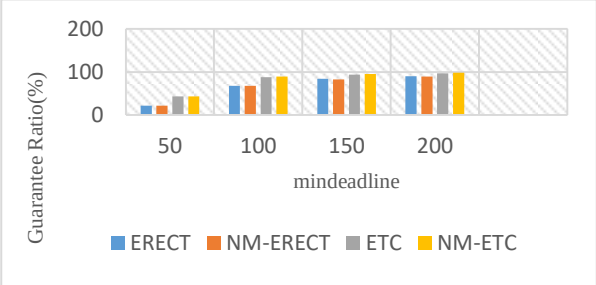

Figure 12. Deadline hit ratio comparison in different lengths.

Figure 13 compares the resource utilization rates. As shown in this figure, the ETC algorithm has improved it by about 26.25% on average in comparison to ERECT. Also NM-ETC has improved in comparison to NM-ERECT about 39.25% on average.

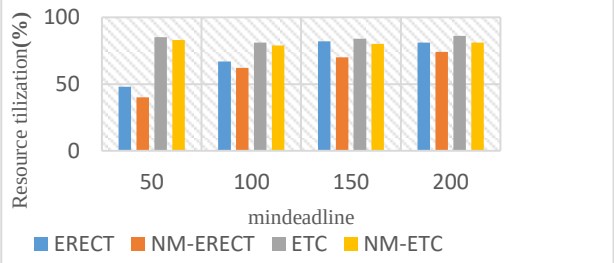

Figure 13. Resource utilization rate comparison.

As shown in Figure 14, the ETC algorithm has improved the energy consumption rate of the accepted tasks by about 16.75% in comparison to ERECT. Also NM-ETC has improved on average about 15.5% in comparison to NM-ERECT.

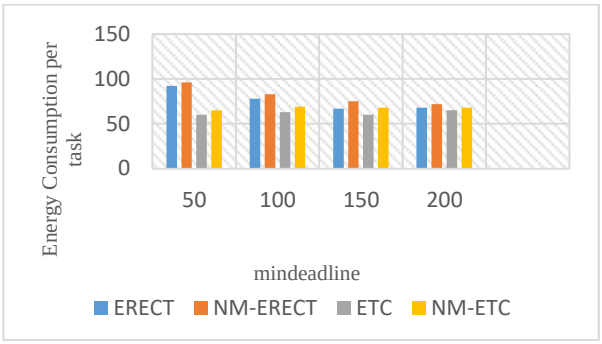

Figure 14. Energy consumption of accepted tasks.

As shown in this section (ETC), since we employed vertical scalability, in a shorter deadline, due to the fact that it solves the effect of start-up time and creates a virtual machine for real-time tasks, it performs much better in terms of the guarantee ratio, energy consumption, and resource utilization.

8. Conclusions

In this paper, we proposed the energy-aware real-time task (EaRTs) scheduling approach in a cloud computing environment. In order to guarantee the real-time tasks, the vertical scaling of VMs was used when the horizontal scaling could not guarantee the real-time task. Our approach included the four algorithms ETC, HVS, V2S, and PSD. Our proposed task scheduling approach was modelled formally in timed automata and the properties, namely reachability, deadlock-freeness, and correctness, were analyzed and proved using the UPPAAL model checker. We checked the schedulability of EaRTs through the reachability property, and showed that our approach guaranteed that the deadlines of all tasks were met. Our experimental results showed higher remaining energies and resource utilization.

References

[1] M. Kumar, S.C. Sharma, A. Goel, A. and S.P. Singh, "A comprehensive survey for scheduling techniques in cloud computing" *Journal of Network and Computer Applications*, vol. 143, pp. 1-33, October 2019.

[2] M. Tajamolian and M. Ghasemzadeh, "Analytical evaluation of an innovative decision-making algorithm for VM live migration" *Journal of AI and Data Mining*, vol. 7, no. 4, pp. 589-596, November 2019.

[3] A. Amini Motlagh, A. Movaghar and A. M. Rahmani, "Task scheduling mechanisms in cloud computing: A systematic review" *International Journal of Communication Systems*, vol. 33, no. 6, pp. 1-23, April 2020.

[4] Y. Saadi and S. El Kafhali, "Energy-efficient strategy for virtual machine consolidation in cloud environment" *Soft Computing*, pp. 1-15, March 2020.

[5] H. Chen, X. Zhu, J. Zhu and J. Wang, "Eres: An energy-aware real-time elastic scheduling algorithm in clouds". *In IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, 2013, pp. 777-784.

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

[7] G. Chen, N. Guan, K. Huang, and W. Yi, "Fault-tolerant real-time tasks scheduling with dynamic fault handling". *Journal of Systems Architecture*, vol. 102, January 2020.

[8] S. Wimmer and J. Mutius, "Verified certification of reachability checking for timed automata". *In International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2020, pp. 425-443.

[9] J. Singh, "Schedulability Analysis of Probabilistic Real-Time Systems" Doctoral dissertation, UNIVERSITE DE TOULOUSE. 2020.

[10] B. Keshanchi, A. Souri, and N.J. Navimipour, "An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing" *Journal of Systems and Software*, vol.124, pp. 1-21, February 2017.

[11] H. Chen, G. Liu, S. Yin, X. Liu and D. Qiu, "ERECT: Energy-efficient reactive scheduling for real-time tasks in heterogeneous virtualized clouds" *Journal of computational science*, vol. 28, pp. 416-425, September 2018.

[12] Z. Deng, G. Zeng, Q. He, Y. Zhong, and W. Wang, "Using priced timed automaton to analyse the energy consumption in cloud computing environment" *Cluster computing*, vol.17, no. 4, pp. 1295-1307, December 2014.

[13] N. Akhter and M. Othman, "Energy aware resource allocation of cloud data centre: review and open issues" *Cluster Computing*, vol. 19, no. 3, pp. 1163-1182, September 2016.

[14] A.A.S. Ahmad and P. Andras, "Scalability analysis comparisons of cloud-based software services" *Journal of Cloud Computing*, vol. 8, no. 1, pp. 1-17, December 2019.

[15] J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu and J. Chen, "A cost-aware auto-scaling approach using the workload prediction in service clouds" *Information Systems Frontiers*, vol.16, no.1, pp.7-18, March 2014.

[16] A. David, J. Illum, K.G. Larsen and A. Skou, "Model-based framework for schedulability analysis using UPPAAL 4.1." *Model-based design for embedded systems*, vol.1, no.1, pp. 93-119, January 2009.

[17] M. Mikučionis, K.G. Larsen, J.I. Rasmussen, B. Nielsen, A. Skou, S.U. Palm and P. Hougaard, "Schedulability analysis using Uppaal: Herschel-Planck case study" *In International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, Springer, Berlin, *Heidelberg*, 2010, pp. 175-190.

[18] N. Saeedloei and F. Kluźniak, "Synthesizing clock-efficient timed automata" *In International Conference on Integrated Formal Methods*, Springer, Cham, 2020, pp. 276-294.

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

[20] Y. Jun, M. Qingqiang, W. Song, L. Duanchao, H. Taigui and D. Wanchun, "Energy-aware tasks scheduling with deadline-constrained in clouds" I*n IEEE International Conference on Advanced Cloud and Big Data (CBD)*, 2016, pp. 116-12.

[21] Y. Zhang, L. Chen, H. Shen and X. Cheng, "An energy-efficient task scheduling heuristic algorithm without virtual machine migration in real-time cloud environments". *In International Conference on Network and System Security*, Springer, Cham, 2016, pp. 80-97.

[22] S. Hosseinimotlagh, F. Khunjush and R. Samadzadeh, "Seats: smart energy-aware task scheduling in real-time cloud computing" *The Journal of Supercomputing*, vol. 71, no. 1, pp. 45-66, January 2015.

[23] J. Wang, W. Bao, X. Zhu, L.T. Yang and Y. Xiang, "FESTAL: fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds" *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2545-2558, November 2014.

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

[25] K.G. Larsen, P. Pettersson and W. Yi, "UPPAAL in a nutshell" *International journal on software tools for technology transfer*, vol. 1, no 1-2, pp. 134-152, December 1997.

[26] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson and W. Yi, "UPPAAL—a tool suite for automatic verification of real-time systems. *In International hybrid systems workshop*, Springer, Berlin, Heidelberg, October 1995, pp. 232-243.

[27] Y.A.K. Chaudhry and M. Hammed, "Formal Verification of Cloud based Distributed System using UPPAAL" *In IEEE International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies*, 2019, pp. 1-4.

[28] G. Behrmann, A. David and K.G. Larsen, "A tutorial on uppaal" *In Formal methods for the design of real-time systems*, Springer, Berlin, Heidelberg, September 2004, pp. 200-236.

[29] Fersman, E., Mokrushin, L., Pettersson, P., & Yi, W. (2006). Schedulability analysis of fixed-priority systems using timed automata. Theoretical Computer Science, Vol. 354, No. 2, pp. 301-317.

[30] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A. De Rose and R. Buyya, "CloudSim: a toolkit for modelling and simulation of cloud computing environments and evaluation of resource provisioning algorithms" *Software: Practice and experience*, vol. 41, no. 1, pp. 23-50, January 2011.

راهکار زمانبندی وظایف بی‌درنگ آگاه از انرژی در یک محیط محاسبات ابری

ناهید مبهوت و حسین مومنی*

گروه آموزشی مهندسی کامپیوتر، دانشگاه گلستان، گرگان، ایران.

ارسال ۲۰۲۰/۱۲/۰۹؛ بازنگری ۲۰۲۱/۰۲/۱۵؛ پذیرش ۲۰۲۱/۰۳/۰۶

چکیده:

انگیزه استفاده از رایانش ابری طی سالهای اخیر بطور قابل توجهی افزایش یافته است که دلیل اصلی آن مقیاس پذیری منابع مجازی است. رایانش ابری به پیشرفت برنامه‌های بی‌درنگ مانند پردازش سیگنال، نظارت محیطی و پیش‌بینی هوا که در آن ملاحظات زمان و انرژی برای انجام وظایف بسیار مهم است، کمک کرده است. در برنامه‌های بی‌درنگ، از دست دادن مهلت زمانی وظایف عواقب فاجعه باری در پی دارد بنابراین زمانبندی وظایف بی‌درنگ یک مسئله مهم و اساسی است. علاوه بر این، صرفه جویی در انرژی مراکز داده ابری، با توجه به مزایایی مانند کاهش هزینه های عملیاتی و حفاظت از محیط زیست، مسئله مهمی است که طی سالهای اخیر مورد توجه قرار گرفته است و با برنامه ریزی مناسب کار قابل کاهش است. در این مقاله، ما یک رویکرد زمانبندی بی‌درنگ وظیفه آگاه از انرژی(EaRT) را برای کاربردهای بی‌درنگ ارائه می دهیم. ما از تکنیک‌های مجازی سازی و تلفیق استفاده می‌کنیم که این امر موجب به حداقل رساندن مصرف انرژی، بهبود استفاده از منابع و رعایت موعد زمانی وظایف می‌گردد. در روش تلفیق، افزایش و کاهش تعداد منابع مجازی سازی می تواند عملکرد اجرای کار را بهبود بخشد. در روش پیشنهادی چهار الگوریتم ارایه خواهیم داد و برای اثبات دقیق ویژگی زمانبندی و درستی راهکار پیشنهادی از روش مدل رسمی و به کمک آتوماتای زمانی استفاده خواهیم نمود. نتایج ارزیابی نشان می‌دهد که روش پیشنهادی ما در معیارهای موعد زمانی، استفاده از منابع و مصرف انرژی در مقایسه با سایر الگوریتم های زمانبندی وظایف بی‌درنگ آگاه از انرژی کارآمدتر است.

کلمات کلیدی: وظیفه، بی‌درنگ، محاسبات ابری، مقیاس بالا، مقایس پایین، زمانبندی.