Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 50 (2015) 176 – 184

Procedia Computer Science

2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)

Allocation-Aware Task Scheduling for Heterogeneous Multi-Cloud Systems

Sanjaya K. Panda[a], *IEEE Member*, Indrajeet Gupta[b] and Prasanta K. Jana[c], *IEEE Senior Member*

[a] *Department of Information Technology*
Veer Surendra Sai University of Technology, Burla – 768018, India
[b,c] *Department of Computer Science and Engineering*
Indian School of Mines, Dhanbad – 826004, India

Abstract

Cloud computing is one of the growing technology usage for the day-to-day business operations in today's IT industry. The diverse features of cloud such as on-demand self-service, quality of service, pay-per-usage pricing, virtualization and elasticity make the cloud more popular in industries as well as research communities. However, the mapping of the cloud resources in forms of virtual machines (VMs) to fulfill the customer requests is very challenging and a well-known NP-Complete problem. In this paper, we propose an allocation-aware task scheduling (ATS) algorithm for heterogeneous multi-cloud systems. The algorithm has three phases, namely matching, allocating and scheduling that aim to map the customer requests (or tasks) to the VMs of the clouds such that the overall completion time i.e., makespan is minimized. Moreover, the algorithm introduces a new phase called allocating to reschedule the tasks to meet the requirement of scheduling strategy. We perform rigorous experiments on benchmark as well as synthetic datasets and compare the experimental results by extending two existing multi-cloud scheduling algorithms as per the proposed model. The results clearly indicate that the proposed algorithm outperforms both the algorithms in terms of makespan and average cloud utilization.

Peer-review under responsibility of scientific committee of 2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)

Keywords: Task Allocation; Multi-Cloud; Virtual Machine; Scheduler; Cloud Manager

1. Introduction

Cloud computing provides on-demand self-services to the customers on pay-per-usage basis. The cloud services are provisioned in the form of storage, computation, network and many more. IaaS (Infrastructure as a Service) cloud provides the services by deploying virtual machines (VMs) in their data center. However, IaaS cloud uses a scheduling policy to allocate VMs to the customer requests. For instance, GoGrid [1] uses round robin (RR)

Peer-review under responsibility of scientific committee of 2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)
doi:10.1016/j.procs.2015.04.081

scheduling to assign customer requests to the VMs evenly. OpenNebula [2] uses rank scheduling policy to the services that are more suitable for VMs. Least connect scheduling [3] assigns the requests to the VMs that are least loaded. Weighted least scheduling connect [4] assigns a weight value to each VM. The VM with higher weight is assigned large number of requests at a time. Thus the future of cloud computing will collaborate among the cloud service providers (CSPs) to solve the complex problems dealing with scientific and engineering applications. It is essential because there is no data center which has unlimited resources to handle such complex problems.

The inter-cloud collaboration makes the task scheduling very challenging and is the main focus of current research in cloud computing. In this case, the CSPs may accept the centralized management to perform task scheduling. The centralized node keeps track of the current status of the VMs and the scheduling policy in each cloud. Alternatively, assigning the requests to the VMs or clouds without concerning the scheduling policy makes task scheduling very inefficient. Moreover, mapping the requests to the VMs is a two-phase process, namely matching followed by scheduling. The matching finds the VMs for the tasks whereas scheduling arranges the execution order of all the tasks. However, there is a gap between the two phases as far as multi-cloud collaboration is considered. The task-cloud pair in the matching phase may differ from the task-cloud pair in the scheduling phase. This is due to CSPs that are scheduled using different scheduling policy.

In this paper, we propose a new algorithm called allocation-aware task scheduling (ATS). The algorithm is a three phase process consisting of matching, allocating and scheduling. The allocating phase fulfills the gap between matching and scheduling. The algorithm is tested with synthetic and benchmark datasets. The experimental results show that the proposed model along with the proposed algorithm performs better than the existing algorithms in terms of makespan (i.e., the overall completion time to execute all the tasks) and average cloud utilization. Our major contribution can be summarized as follows. 1) Development of a task scheduling algorithm for collaboration of multiple CSP in heterogeneous environment. 2) Present a new phase called allocating to fulfill the gap between matching and scheduling. 3) Simulation of the proposed algorithm in benchmark and synthetic datasets. 4) Comparison of the experimental results with the existing algorithms.

The remainder of this paper is organized as follows. Section 2 presents related work with their applicability in multi-cloud collaboration. Section 3 explains the proposed cloud model and the scheduling problem associated with the model. We present the proposed algorithm in Section 4 followed by the experimental results in Section 5. We conclude the paper in Section 6.

2. Related Work

There have been a number of algorithms for task scheduling in cloud computing. They map the customer requests to the clouds by assuming the VMs within a data center form a cloud [5]. Chen et al. [6] have proposed user-priority guided min-min scheduling algorithm which is an extension of the traditional min-min [7-8] algorithm. However, the customer requests (or tasks) are mapped to the resources (or clouds) instead of VMs which is created under the clouds. Similarly Kim et al. [9] have presented biogeography-based optimization (BBO) for job scheduling. The performance of BBO is better than other optimization problems such as genetic algorithm (GA), particle swarm optimization (PSO) and simulated annealing (SA) for large size problems. Here also the jobs are scheduled to the clouds instead of VMs. GoGrid [1] cloud used RR algorithm to distribute the tasks evenly. However, due to task heterogeneity, the algorithm does not produce better makespan. Recently, Li et al. [5] have presented two algorithms, namely cloud list scheduling (CLS) and cloud min-min scheduling (CMMS) for heterogeneous multi-cloud environment. CLS is a single-phase scheduling and CMMS is a two-phase scheduling. However, both scheduling algorithms assign the tasks of an application to the clouds without concerning the VMs creation.

3. Cloud Model and Problem Statement

3.1. *Cloud Model*

Consider a multi-cloud system in which each cloud has its own data center consisting of a set of servers to deploy the VMs [5]. Each CSP has its own scheduler that assigns the tasks to the VMs using its own scheduling strategy. Note that the scheduling strategy of the clouds may be different. For example, Amazon Elastic Compute Cloud (EC2) and Microsoft Azure use different scheduling strategies. The future cloud computing may have multiple cloud providers to run various applications [10-11]. In that case, the service providers may adopt a centralized

management approach to achieve some common goals [5], [12-13]. The proposed cloud model has following key components:

1. **Customer:** Customers are the service consumers of the clouds. They place the service request with the help cloud manager.

2. **Cloud Manager:** Cloud manager receives the service request from the customers. On the other hand, it keeps track of number of active VMs on each cloud. Furthermore, the cloud manager tracks the scheduling strategy used by each cloud. Note that, VMs are heterogeneous in nature as they have different computing capabilities and specifications like storage, network etc.

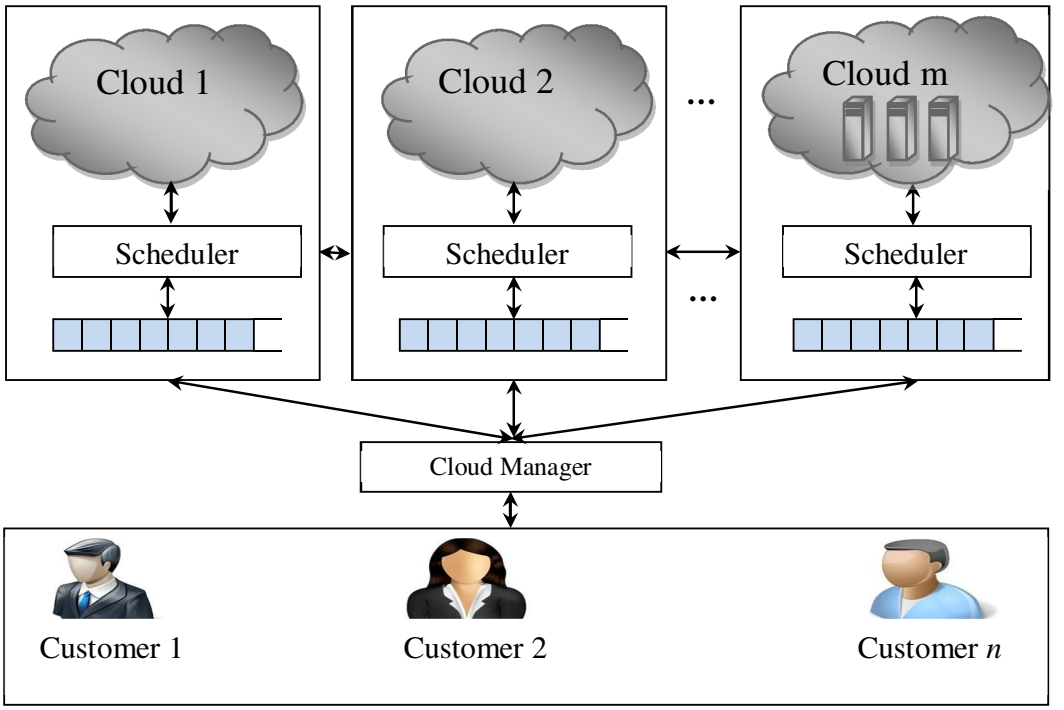

Fig. 1. The Proposed Cloud Model

3. **Cloud Service Provider (CSP):** CSPs are the producers of the clouds. They provide the services by deploying VMs on the active servers. These VMs execute the service requests as per the scheduling strategy. However, the scheduling strategies of the service providers are not be same or different as stated earlier.

3.2. *Problem Statement*

***Definition* 1.** Let $C = \{C_1, C_2, C_3, \ldots, C_m\}$ denote the set of clouds and let $C_i = \{S_i, |V_i|\}$ denote the i^{th} cloud with scheduling algorithm (S_i) and the number of active VMs ($|V_i|$). The scheduling algorithm S_i is used to schedule the tasks assigned to the cloud C_i. The active VMs are used to execute the assigned tasks on the clouds. Note that S_i and $|V_i|$ may vary from one cloud to another cloud, e. g., $C_1 = \{RR, 4\}$, $C_2 = \{CLS, 3\}$ and $C_3 = \{CLS, 5\}$ indicate that C_1 has round robin (RR) scheduling strategy with 4 active VMs and C_2 and C_3 have cloud list scheduling (CLS) with 3 and 5 active VMs respectively.

***Definition* 2.** Let $T = \{T_0, T_1, T_2, \ldots, T_p\}$ denote the set of tasks which are to be assigned to the clouds. Let $V_j = \{P, B\}$ denote the j^{th} virtual machine with the processing speed P in MIPS and bandwidth B in Mbps. Similarly let $T_i = \{I, D\}$ denote the i^{th} task with the set of instruction I in MI and data D in Mb. Then the execution time of T_i on V_j is expressed as follows:

$$ETC_{ij} = \frac{I}{P} + \frac{D}{B} \tag{1}$$

For example, the execution time required to execute a task $T_1 = \{200, 20\}$ on a VM $V_1 = \{100, 10\}$ is 3 time units.

***Definition* 3.** A cloud manager ℂ receives a request set $R = \{R_1, R_2,\ldots, R_q\}$ from a customer set $CU = \{CU_1, CU_2, CU_3,\ldots, CU_q\}$ where a service request $R_i = \{T_0, T_1,\ldots, T_p\}$ denote a set of tasks given by a customer CU_i. The cloud manager then maps this request set with the set of clouds $C = \{C_1, C_2, C_3,\ldots, C_m\}$. It keeps track of all the active VMs on a particular cloud.

The problem is to map the customer requests (or tasks) to the clouds such that the following objective is fulfilled.

***Objective* 1.** *The overall processing time* (*makespan*) *is minimized*.

***Objective* 2.** *Each cloud must receive those tasks which results least makespan in its VMs*.

Note that we call this scheduling as "Allocation-aware task scheduling" as the manager sends the tasks to the clouds based on their scheduling algorithm (Objective 2) that results minimization of makespan (Objective 1 and Objective 2).

4. Proposed Algorithm

The basic idea of the proposed algorithm is as follows. It consists of three-phases. In the first phase called matching, the cloud manager maintains a global queue (Q_g) to place the incoming service requests from the customers. Here, the service requests are in the form of one or more tasks. Then the cloud manager removes a task (say, task i) from the global queue and calculates the completion time of task i on each VM of the CSPs. Note that completion time is the sum of the execution time and the ready time. Furthermore, the manager finds the VM that holds minimum completion time for task i (say, VM j) followed by the cloud that holds the VM (say, cloud k). This completes the first phase. In the second phase called allocating, the manager finds the scheduling strategy of cloud k. The rationale behind this is that 1) the service requests are provisioned as per the scheduling strategy 2) the service requests need to be assigned such that makespan is minimized. For instance, the scheduling strategy is RR on cloud k with three VMs and task 1 achieves minimum completion time on VM 2. Then the manager will not assign the task 1 to VM 2 because the cloud k is going to assign VM 1 for task 1 that is first in first out (FIFO) order. Consequently, the manager will find another VM for task 1 that gives minimum completion time. In contrary, if the scheduling strategy is CLS then task 1 is scheduled on VM 2 because CLS is not assigned in FIFO order. This phase may alter the assignment as per scheduling strategy. In the third phase, task scheduling is performed to carry out the computations. Note that each cloud can execute more than one task simultaneously. We use the following terminologies for the pseudo code of the algorithm shown in Fig. 2.

Notation	Definition
Q_g	Global queue of all the tasks
$ETC(i, j)$	Expected time to compute task i on cloud j
$RT(j)$	Ready time of cloud j
$CT(i, j)$	Completion time of task i on cloud j
n	Total number of VMs in all the clouds
M	Total number of clouds
$vmcount_k$	Total number of VMs on cloud k

The proposed algorithm places the service requests in the global queue (Line 1). The requests are served in first-come first-serve order. Thus the manager selects one task from the queue at a time (Line 2). The manager matches the task with all the VMs (Lines 3 and 5) to find the best VM over all (Lines 6 to 13). Then the manager finds the cloud k where the best VM (say, *index*) is deployed (Lines 14 to 15). To know the status of the best VM, Procedure 1 is called. It checks the scheduling strategy of cloud k i.e., RR or CLS in the proposed algorithm. If the scheduling strategy is RR then it finds *vmcount* of cloud k (represented as $vmcount_k$) (Lines 1-2 of Procedure 1). The $vmcount_k$ value ranges from the first VM to the last VM in the cloud k. Initially, the value of $vmcount_k = 1$. Once the first VM receives the task the $vmcount_k$ is updated to 2 and so on. The reason behind this is that RR schedules the first VM followed by second VM and so on. If the $vmcount_k$ is the same as *index*, then the task is assigned to *index* VM (Lines 3-4). Otherwise, it returns assignment failure (Lines 8-9). On the other hand, if the scheduling strategy is CLS then it directly assigns the task to *index* VM (Lines 11-15 of Procedure 1). If the task is not schedulable by RR and cloud k does not follow CLS scheduling strategy, then it calls the Procedure 2. This procedure assigns infinity value (or a very big number) to the previous assignment as it was failed in the last assignment (Lines 1 of Procedure 2). Then it finds the best VM over all the clouds (Lines 2-9). At last, it calls the Procedure 1 to assign the task to the

best VM (Line 12). The process is repeated until the task is scheduled to the best VM and the scheduling strategy is assigning the task to the best VM (Lines 17-19 of main algorithm).

Algorithm: ATS

```
1. while Q_g ≠ NULL do
2.     i ← Delete(Q_g)
3.     for j = 1, 2, 3, … , n
4.         CT(i, j) = ETC(i, j) + RT(j)
5.     endfor
6.     minimum = CT(1, 1)
7.     index = 1
8.     for j = 2, 3, 4, … , n
9.         if minimum > CT(i, j)
10.            minimum = CT(i, j)
11.            index = j
12.        endif
13.    endfor
14.    for k = 1, 2, 3, …, M
15.        if index virtual machine is
           deployed under a cloud k
16.            Call SCHEDULE
               (i, k, index)
17.            while the task is not
               schedulable
18.                Call RESCHEDULE
                   (i, k, index)
19.            endwhile
20.        endif
21.    endfor
22. endwhile
```

Procedure 1: *SCHEDULE*(*i*, *k*, *index*)

```
1.  if cloud k schedules by RR algorithm
2.      Find vmcount_k
3.      if vmcount_k = index
4.          Assign task i to index virtual machine
5.          RT(index) = RT(index) + ETC(i, index)
6.          Remove task i from Q_g
7.          Update vmcount_k
8.      else
9.          Return Failure
10.     endif
11. else cloud k schedules by CLS algorithm
12.     Assign task i to index virtual machine
13.     RT(index) = RT(index) + ETC(i, index)
14.     Remove task i from Q_g
15. endif
16. Return
```

Procedure 2: *RESCHEDULE*(*i*, *k*, *index*)

```
1.  CT(i, index) = ∞
2.  minimum = CT(1, 1)
3.  index = 1
4.  for j = 2, 3, 4, … , n
5.      if minimum > CT(i, j)
6.          minimum = CT(i, j)
7.          index = j
8.      endif
9.  endfor
10. for k = 1, 2, 3, …, M
11.     if index virtual machine is deployed under a
        cloud k
12.         Call SCHEDULE(i, k, index)
13.     endif
14. endfor
```

Fig. 2. Pseudo code for ATS

4.1. *An Illustration*

Let us consider the ETC matrix as shown in Table 1. There are two different clouds and each cloud contains two VMs with their scheduling strategy RR and CLS respectively. We assume that the tasks are arrived to the cloud manager in numeric order. When task T_0 arrives, the manager finds the VM that takes minimum completion time i.e., VM_2 of cloud 1. However, when manager dispatches task T_0, it is scheduled VM_1 as per RR scheduling strategy. Therefore, the matching is rejected and the corresponding ETC value is replaced by ∞ i.e., $ETC(T_0, VM_2) = \infty$. Thus the manager finds another VM that takes minimum completion time for task T_0 which results VM_3. So, the task T_0 is scheduled to VM_3. Next task T_1 is scheduled to VM_1 as it takes minimum completion time and RR scheduling is assigned T_1 to VM_1. Like task T_1, task T_2 is scheduled to VM_2. Then task T_3 is scheduled to VM_4 because the manager found that the minimum completion time for task T_3 in all the VMs i.e., 6 + 2, 3 + 2, 8 + 3 and 5 + 0 respectively in which 5 + 0 is the minimum. Next for task T_4, the minimum completion time in all the VMs is 4 + 2, 7 + 2, 5 + 3 and 3 + 5 respectively. Therefore task T_4 is scheduled to be scheduled to VM_1 as 4 + 2 is the minimum one and so on.

The task-VM mapping is shown in Fig. 3.

Table 1. The ETC matrix for 10 tasks and 4 VMs

		T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
Cloud 1 (RR)	VM_1	9	2	7	6	4	5	9	10	8	7	4
	VM_2	2	5	2	3	7	6	8	6	5	4	9
Cloud 2 (CLS)	VM_3	3	10	9	8	5	3	4	5	7	3	5
	VM_4	8	6	3	5	3	8	3	4	2	9	6

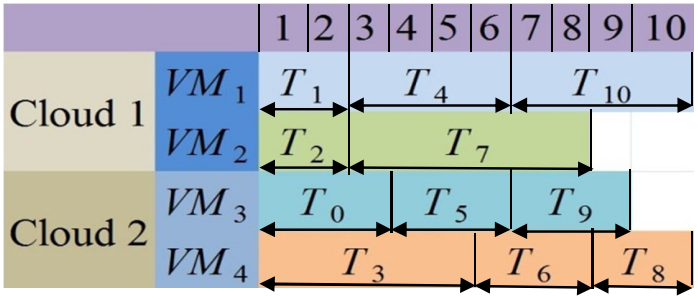

Fig. 3. Gantt chart for the proposed algorithm

Theorem* 4.1: *The overall time complexity of the algorithm ATS is O(ln).

***Proof*:** Let l be the total number of tasks and n is the total number of VMs. Line 2 requires $O(1)$ time. To find the completion time, Line 3 to 5 requires $O(n)$ time. Line 6 to 7 require $O(1)$ time followed by Line 8 to 13 require $O(n)$ time. Line 15 to 21 require $O(n)$ time. The Procedure *SCHEDULE* requires $O(1)$ time and the Procedure *RESCHEDULE* requires $O(n\text{-}1)$ time. So, the overall time complexity of ATS is $O(ln)$ time for l tasks.

Theorem* 4.2: *The Procedure RESCHEDULE requires no more than O((n/2)-1) time iff (n/2) VMs are used in each scheduling strategy (i.e., RR and CLS).

***Proof*:** Let $\{VM_1, VM_2,\ldots, VM_{n/2}\}$ uses RR scheduling strategy and $\{VM_{(n/2)+1}, VM_{(n/2)+2},\ldots, VM_n\}$ uses CLS scheduling strategy. In the worst case, the T_1 is scheduled to VM_i, $2 \le i \le n/2$ and $|i| = ((n/2)\text{-}1)$. The first *RESCHEDULE* Procedure is called to find another VM_j, $2 \le j \le n/2$, $i \ne j$ and $|j| = ((n/2)\text{-}2)$. The last *RESCHEDULE* Procedure is called to find another VM_k, $2 \le k \le n/2$, $i \ne k$, $j \ne k$ and $|k| = ((n/2)\text{-}(n/2))$. The total number of Procedure call is $(n/2)\text{-}2+1 = (O(n/2)\text{-}1)$.

5. Experimental Results

We evaluate the proposed algorithm through simulation run with some benchmark and synthetic datasets. The experiments were carried out using MATLAB R2012a version 7.14.0.739 on an Intel Core 2 Duo processor, 2.20 GHz CPU and 4 GB RAM running on Microsoft Windows 7 platform.

5.1. *Performance Metrics*

5.1.1. *Makespan*

It is the overall completion time needed to assign all the tasks to the available clouds. Let $T_i \rightarrow C_j$ denote the task T_i is assigned to cloud C_j. Then it is mathematically defined as follows

$$M = \max\left(\sum_{i=1}^{x} ETC(i,1)\times F(i,1),\quad \sum_{i=1}^{x} ETC(i,2)\times F(i,2),\quad \ldots\quad ,\sum_{i=1}^{x} ETC(i,m)\times F(i,m)\right) \tag{2}$$

$$\text{where } F(i,j)=\begin{cases}1 & if\ T_i \rightarrow C_j \\ 0 & Otherwise\end{cases} \tag{3}$$

5.1.2. ***Average Cloud Utilization***

It is the average amount of time in which the cloud resources are busy. It ranges from 0 to 1. It is defined by

$$U=\frac{\sum_{i=1}^{m}\frac{M(C_i)}{M}}{m} \tag{4}$$

where $M(C_i)$ denotes the makespan of cloud C

5.2. ***Datasets***

We picked two benchmark datasets generated by Braun et al. [14] and one synthetic dataset formed by MATLAB pseudorandom integer function. The first benchmark dataset consists of 512 tasks to be scheduled to 16 VMs which contains 12 different instances. The instances are denoted by *u_x_yyzz* where *u* is the uniform distribution used to generate the instances, *x* is the type of consistency (i.e., consistent (*c*), inconsistent (*i*) or semi-consistent (*s*)) and *yyzz* is the task and VM heterogeneity (i.e., *hihi*, *hilo*, *lohi*, *lolo*) respectively. Similarly, the second dataset contains 1024 tasks which are assigned to 32 different VMs. We denote the datasets 512 × 16 and 1024 × 32 respectively. We assume that there are four clouds in 512 × 16 dataset in which each cloud holds four VMs. In the similar fashion, eight clouds hold 32 VMs in 1024 × 32 dataset. These datasets are used in task scheduling [15-23]. The synthetic dataset consists of six instances. The instances are represented by $v \times w$ where v is the total number of tasks that are assigned w clouds. Note that the VMs in each cloud are fixed for the simplicity of experiments. However, there may be any number of VMs created in each cloud based on their capacity and the manager must keep track of the number of active VMs in each cloud.

5.3. ***Experimental Results***

We divide the experimental results in to two parts. In the first part, the scheduling strategy of all the clouds is same. For instance, if one cloud schedules the tasks using RR scheduling strategy then other clouds are also schedule the tasks using RR scheduling strategy. On the other hand, the scheduling strategy of each cloud is different in the second part. For instance, cloud 1 schedules the tasks using RR whereas cloud 2 uses CLS scheduling and so on. Alternatively, we assume that one of the two scheduling algorithms, RR or CLS is used by each cloud or all clouds and four VMs are active in each cloud. We call the proposed algorithm as ATS-RR when all the clouds schedule the tasks using RR scheduling strategy and ATS-RR-CLS when RR and CLS scheduling strategies are used to schedule the tasks. The comparison of makespan and average cloud utilization of RR, ATS-RR, CLS and ATS-CLS for 512 × 16 and 1024 × 32 benchmark datasets are shown in Table 2 and Table 3 respectively.

Table 2. Comparison of makespan for RR, ATS-RR, CLS and ATS-CLS using benchmark datasets

Instance	RR (512 × 16)	ATS-RR (512 × 16)	CLS (512 × 16)	ATS-CLS (512 × 16)	RR (1024 × 32)	ATS-RR (1024 × 32)	CLS (1024 × 32)	ATS-CLS (1024 × 32)
u_c_hihi	4.6589e+07	2.2930e+07	4.7472e+07	1.1423e+07	1.6070e+08	5.9123e+07	1.5447e+08	3.2833e+07
u_c_hilo	4.8937e+05	2.7474e+05	1.1851e+06	1.8589e+05	1.6570e+07	6.2427e+06	1.5504e+07	3.2458e+06
u_c_lohi	1.5174e+06	7.9683e+05	1.4531e+06	3.7830e+05	1.9752e+04	6.1469e+03	1.4151e+04	3.0587e+03
u_c_lolo	1.8132e+04	9.7440e+03	3.9582e+04	6.3601e+03	1.4582e+03	6.7273e+02	1.5675e+03	3.2628e+02
u_i_hihi	3.4409e+07	1.6272e+07	4.5085e+06	4.4136e+06	1.0113e+08	3.1518e+07	7.4620e+06	7.5671e+06
u_i_hilo	3.0365e+05	2.0020e+05	9.6610e+04	9.4856e+04	9.6486e+06	3.4381e+06	7.6598e+05	7.1313e+05
u_i_lohi	1.0015e+06	5.3748e+05	1.8569e+05	1.4382e+05	9.9151e+03	3.5443e+03	8.5439e+02	7.5410e+02
u_i_lolo	1.1034e+04	7.4452e+03	3.3993e+03	3.1374e+03	1.0733e+03	3.6450e+02	9.1120e+01	7.2390e+01
u_s_hihi	4.1530e+07	2.1845e+07	2.5162e+07	6.6939e+06	1.5160e+08	4.7943e+07	8.4821e+07	1.9008e+07
u_s_hilo	4.4659e+05	2.5367e+05	6.0536e+05	1.2659e+05	1.6722e+07	4.2095e+06	8.0988e+06	1.8255e+06
u_s_lohi	1.1974e+06	7.3769e+05	6.7469e+05	1.8615e+05	1.5227e+04	4.0046e+03	8.3377e+03	1.8220e+03
u_s_lolo	1.5397e+04	9.4821e+03	2.1042e+04	4.4361e+03	1.5880e+03	4.4558e+02	8.0161e+02	1.9423e+02

Table 3. Comparison of average cloud utilization for RR, ATS-RR, CLS and ATS-CLS using benchmark datasets

Instance	RR (512 × 16)	ATS-RR (512 × 16)	CLS (512 × 16)	ATS-CLS (512 × 16)	RR (1024 × 32)	ATS-RR (1024 × 32)	CLS (1024 × 32)	ATS-CLS (1024 × 32)
u_c_hihi	0.51	0.80	1	0.95	0.50	0.85	1	0.93
u_c_hilo	0.54	0.84	1	0.97	0.48	0.81	1	0.94
u_c_lohi	0.53	0.79	1	0.96	0.40	0.81	1	0.92
u_c_lolo	0.50	0.79	1	0.95	0.54	0.80	1	0.95
u_i_hihi	0.69	0.93	0.62	0.93	0.81	0.93	0.66	0.91
u_i_hilo	0.84	0.95	0.75	0.95	0.78	0.90	0.60	0.91
u_i_lohi	0.82	0.94	0.53	0.94	0.80	0.89	0.57	0.91
u_i_lolo	0.81	0.92	0.74	0.96	0.72	0.90	0.52	0.91
u_s_hihi	0.57	0.87	0.19	0.92	0.52	0.91	0.05	0.91
u_s_hilo	0.58	0.84	0.21	0.93	0.47	0.92	0.06	0.93
u_s_lohi	0.58	0.76	0.21	0.95	0.50	0.92	0.06	0.89
u_s_lolo	0.61	0.84	0.22	0.95	0.50	0.94	0.06	0.90

Next we compare RR-CLS and ATS-RR-CLS using benchmark datasets. The comparison of both makepsan and average cloud utilization is shown in Table 4.

Table 4. Comparison of makespan and average cloud utilization for RR-CLS and ATS-RR-CLS using benchmark datasets

Instance	RR-CLS (512 × 16)	ATS-RR-CLS (512 × 16)	RR-CLS (1024 × 32)	ATS-RR-CLS (1024 × 32)	RR-CLS (512 × 16)	ATS-RR-CLS (512 × 16)	RR-CLS (1024 × 32)	ATS-RR-CLS (1024 × 32)
u_c_hihi	3.3903e+07	1.9945e+07	1.0065e+08	5.3123e+07	0.3371	0.8677	0.2903	0.8961
u_c_hilo	3.2317e+05	2.5924e+05	9.8061e+06	4.9743e+06	0.5745	0.8554	0.2933	0.8978
u_c_lohi	1.2006e+06	6.7325e+05	1.0416e+04	5.4921e+03	0.3330	0.8600	0.2490	0.8654
u_c_lolo	1.1040e+04	8.9011e+03	1.0160e+03	5.6015e+02	0.5568	0.8518	0.2847	0.8794
u_i_hihi	2.3136e+07	7.5328e+06	8.7800e+07	1.4293e+07	0.4865	0.9162	0.3527	0.9193
u_i_hilo	3.0530e+05	1.3034e+05	9.5388e+06	1.4134e+06	0.5376	0.9494	0.3037	0.8984
u_i_lohi	7.4806e+05	2.5151e+05	7.2647e+03	1.4487e+03	0.5174	0.9349	0.4057	0.9357
u_i_lolo	1.0177e+04	4.5871e+03	7.1349e+02	1.2927e+02	0.5256	0.9458	0.4004	0.9090
u_s_hihi	4.5194e+07	1.1266e+07	7.3232e+07	3.7281e+07	0.3256	0.9016	0.3355	0.9175
u_s_hilo	4.8018e+05	1.6586e+05	6.5326e+06	3.3115e+06	0.4014	0.9463	0.4003	0.9005
u_s_lohi	1.3603e+06	3.4379e+05	5.9709e+03	3.6808e+03	0.3363	0.9212	0.4437	0.8655
u_s_lolo	1.5199e+04	6.0038e+03	6.2923e+02	3.7205e+02	0.4288	0.9186	0.3730	0.8893

Next we experiment the proposed and existing scheduling algorithms using synthetic dataset. The comparison of makespan and average cloud utilization for RR, ATS-RR, CLS and ATS-CLS scheduling are jointly shown in Table 5.

Table 5. Comparison of makespan and average cloud utilization for RR, ATS-RR, CLS and ATS-CLS using synthetic dataset

Instance	RR	ATS-RR	CLS	ATS-CLS	RR	ATS-RR	CLS	ATS-CLS
128 × 16	2990	1967	1667	1324	0.8043	0.9110	0.6082	0.9162
256 × 16	5663	3558	3058	2561	0.8632	0.9285	0.6410	0.9424
384 × 16	8128	5297	4461	3588	0.8798	0.9610	0.6630	0.9720
640 × 32	6848	3637	3243	2731	0.8643	0.9292	0.6856	0.9623
768 × 32	8326	4322	4608	3231	0.8494	0.9539	0.5798	0.9699
896 × 32	9619	4956	4405	3763	0.8602	0.9650	0.7088	0.9728

The comparison of RR-CLS and ATS-RR-CLS using makespan and average cloud utilization performance metrics is shown in Table 6.

Table 6. Comparison of makespan and average cloud utilization for RR-CLS and ATS-RR-CLS using synthetic dataset

Instance	RR-CLS	ATS-RR-CLS	RR-CLS	ATS-RR-CLS
128 × 16	2473	1492	0.6345	0.9186
256 × 16	5039	2887	0.6602	0.9294
384 × 16	7245	4199	0.7100	0.9701
640 × 32	7191	2979	0.5520	0.9709
768 × 32	8511	3558	0.5525	0.9548
896 × 32	8189	4186	0.6577	0.9666

6. Conclusion

We have presented an allocation-aware task scheduling algorithm for heterogeneous multi-cloud environment. The algorithm has been shown to require $O(ln)$ time for l tasks and n VMs. It was experimented rigorously on benchmark and synthetic dataset. The experimental results have been compared with two multi-cloud task scheduling algorithms, namely RR and CLS. The comparison results show the superiority of the proposed algorithm over existing algorithms in terms of two performance metrics, makespan and average cloud utilization.

References

1. B. P. Rimal, E. Choi and I. Lumb, "A Taxonomy and Survey of Cloud Computing Systems", International Joint Conference on INC, IMS and IDC, pp. 44–51, 2009.
2. Open Nebula, http://archives.opennebula.org/start, Accessed on 28th December 2014.
3. Least Connect, https://wiki.gogrid.com/index.php/(F5)_Load_Balancer, Accessed on 28th December 2014.
4. Weighted Least Connection Scheduling, http://kb.linuxvirtualserver.org/wiki/Weighted_ Least-Connection_Scheduling, Accessed on 28th December 2014.
5. J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin and Z. Gu, "Online Optimization for Scheduling Preemptable Tasks on IaaS Cloud System", Journal of Parallel Distributed Computing, Elsevier, Vol. 72, pp. 666-677, 2012.
6. H. Chen, F. Wang, N. Helian and G. Akanmu, "User-Priority Guided Min-Min Scheduling Algorithm for Load Balancing in Cloud Computing", National Conference on Parallel Computing Technologies, pp. 1-8, 2013.
7. O. H. Ibarra and C. E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors", Journal of the Association for Computing Machinery, Vol. 24, No. 2, pp. 280-289, 1977.
8. R. Armstrong, D. Hensgen and T. Kidd, "The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-time Predictions", 7th IEEE Heterogeneous Computing Workshop, pp. 79-87, 1998.
9. S. Kim, J. Byeon, H. Yu and H. Liu, "Biogeography-Based Optimization for Optimal Job Scheduling in Cloud Computing", Applied Mathematics and Computation, Elsevier, Vol. 247, pp. 266-280, 2014.
10. M. Aron, P. Druschel and W. Zwaenepoel, "Cluster Reserves: A Mechanism for Resource Management in Cluster-Based Network Servers", ACM Sigmetrics, 2000.
11. A. I. Avetisyan, R. Campbell, M. T. Gupta, I. Heath, S. Y. Ko, G. R. Ganger, M. A. Kozuch, D. Hallaron, M. Kunze, T. T. Kwan, K. Lai, M. Lyons, D. S. Milojicic, H. Y. Lee, Y. C. Soh, N. K. Ming, J. Luke and H. Namgoong, "Open Cirrus a Global Cloud Computing Testbed", IEEE Computer Society, Vol. 43, Issue 4, pp. 35-43, 2010.
12. R. Buyya, C. Vecchiola and S. T. Selvi, "Mastering Cloud Computing Foundations and Applications Programming", Morgan Kaufmann, Elsevier, 2013.
13. J. Li, M. Qiu, J. W. Niu, Y. Chen and Z. Ming, "Adaptive Resource Allocation for Preemptable Jobs in Cloud Systems", 10th IEEE International Conference on Intelligent Systems Design and Applications, pp. 31-36, 2010.
14. Braun et al., https://code.google.com/p/hcspchc/source/browse/trunk/AE/ ProblemInstances/ HCSP/Braun_et_al /u_c_hihi.0?r=93, Accessed on 20th April 2014.
15. S. K. Panda, P. Agrawal, P. M. Khilar and D. P. Mohapatra, "Skewness-Based Min-Min Max-Min Heuristic for Grid Task Scheduling", 4th International Conference on Advanced Computing and Communication Technologies, IEEE, pp. 282-289, 2014.
16. F. Xhafa, L. Barolli and A. Durresi, "Batch Mode Scheduling in Grid Systems", International Journal Web and Grid Services, Vol. 3, No. 1, pp. 19-37, 2007.
17. F. Xhafa, J. Carretero, L. Barolli and A. Durresi, "Immediate Mode Scheduling in Grid Systems", International Journal Web and Grid Services, Vol. 3, No. 2, pp. 219-236, 2007.
18. S. K. Panda, P. M. Khilar and D. P. Mohapatra, "FTM^2: Fault Tolerant Batch Mode Heuristics in Computational Grid", 10th International Conference on Distributed Computing and Internet Technology, Lecture Notes in Computer Science, Springer, Vol. 8337, pp. 98-104, 2014.
19. S. K. Panda, P. M. Khilar and D. P. Mohapatra, "FTMXT: Fault Tolerant Immediate Mode Heuristics in Computational Grid", International Conference on Informatics and Communication Technologies for Societal Development, Springer, pp. 103-113, 2014.
20. S. K. Panda and P. K. Jana, "Efficient Task Scheduling Algorithms for Heterogeneous Multi-cloud Environment", The Journal of Supercomputing, Springer, 2015.
21. S. K. Panda and P. K. Jana, "A Multi-Objective Task Scheduling Algorithm for Heterogeneous Multi-cloud Environment", IEEE International Conference on Electronic Design, Computer Networks and Automated Verification, 2015.
22. S. K. Panda**,** S. Nag and P. K. Jana, "A Smoothing Based Task Scheduling Algorithm for Heterogeneous Multi-Cloud Environment", 3rd IEEE International Conference on Parallel, Distributed and Grid Computing (PDGC), 2014.
23. S. K. Panda, P. Agrawal and D. P. Mohapatra, "Intermediate Mode Scheduling in Computational Grid", IEEE International Conference on Green Computing, Communication and Electrical Engineering, 2014, pp. 1 - 6.