Journal of Computer Science, 9 (2): 252-263, 2013
ISSN 1549-3636

doi:10.3844/jcssp.2013.252.263 Published Online 9 (2) 2013 (http://www.thescipub.com/jcs.toc)

Comparative Study of Scheduling Al-gorithms in Cloud Computing Environment

Isam Azawi Mohialdeen

College of Information Technology, University Tenaga Nasional, Selangor, Malaysia

Received 2012-12-18, Revised 2013-03-16; Accepted 2013-04-11

ABSTRACT

An essential requirement in cloud computing environment is scheduling the current jobs to be executed with the given constraints. The scheduler should order the jobs in a way where balance between improving the quality of services and at the same time maintaining the efficiency and fairness among the jobs. Thus, evaluating the performance of scheduling algorithms is crucial towards realizing large-scale distributed systems. In spite of the various scheduling algorithms proposed for cloud environment, there is no comprehensive performance study undertaken which provides a unified platform for comparing such algorithms. Comparing these scheduling algorithms from different perspectives is an aspect that needs to be addressed. This pa-per aims at achieving a practical comparison study among four common job scheduling algorithms in cloud computing. These algorithms are Round Rubin (RR), Random Resource Selection, Opportunistic Load Balancing and Minimum Completion Time. These algorithms have been evaluated in terms of their ability to provide quality service for the tasks and guarantee fairness amongst the jobs served. The three metrics for evaluating these job scheduling algorithms are throughput, makespan and the total execution cost. Several experiments with various aims have been accomplished in this comparative study.

Keywords: Cloud Computing, Job Scheduling, Scheduling Algorithm

1. INTRODUCTION

Nowadays, many companies offering services to the customer based on the concept of “pay as a service”, where each customer pays for the services obtained from the provider. The cloud environment provides a different platform by creating a virtual machine that assists users in accomplishing their jobs within a reasonable time and cost-effectively without sacrificing the quality of the services. The huge growth in virtualization and cloud computing technologies reflect the increasing number of jobs that require the services of the virtual machine. Various types of scheduling algorithms have been applied on various data workloads and measured with different performance metrics to evaluate the performance. Most of the scheduling algorithms are developed to accomplish two aims. The first is to improve the quality of services in executing the jobs and provide the expected output on time. The second is to maintain efficiency and fairness for all jobs. **Figure 1** illustrates the proposed cloud frame-work which consists of three tiers, namely, the cloud provider, the internet and the connected clients.

The scheduler should order the jobs in a way where balance between improving the quality of services and at the same time maintaining the efficiency and fairness among the jobs. Thus, evaluating the performance of scheduling algorithms is crucial towards realizing large-scale distributed systems. In spite of the various scheduling algorithms proposed for cloud environment, there is no comprehensive performance study undertaken which provides a unified platform for comparing such algorithms. Comparing these scheduling algorithms in an Infrastructure as a Service (IaaS) of cloud computing from different perspectives is an aspect that needs to be addressed.

There are numerous literatures which propose scheduling algorithms. Some of these proposed algorithms are particularly for serving jobs in a cloud computing environment and some are tailored to fit the cloud environment.

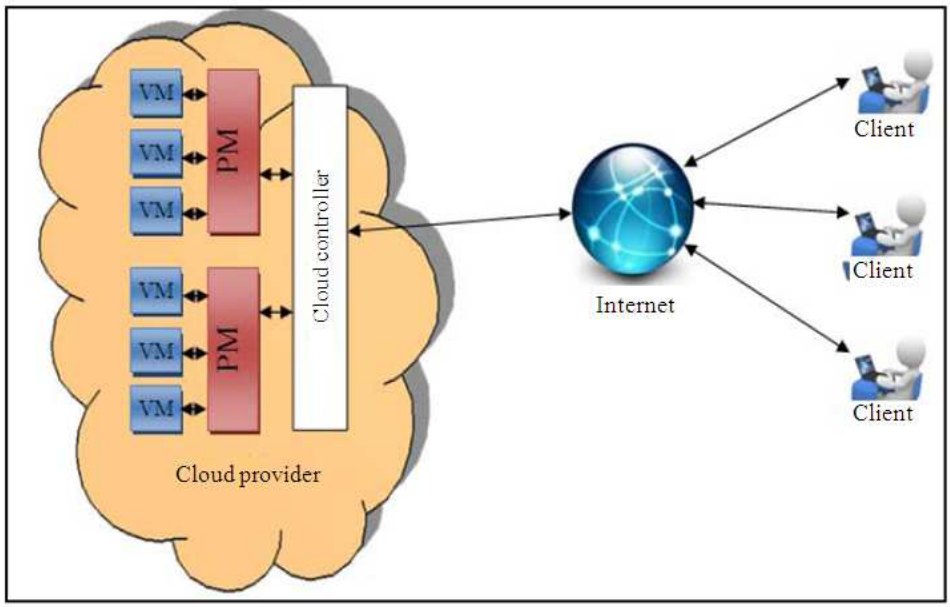

Fig. 1. The proposed cloud framework

For the cloud environment, many adapted scheduling algorithms are proposed to enhance the total system performance such as throughput, make span and the cost. However, the variety of scheduling algorithms increases the complexity of selecting the best one for adoption.

This study aims at analyze and investigate four job scheduling algorithms under cloud environment, namely, Round Robin (RR), Random Resource Selection, Opportunistic Load Balancing and Minimum Completion Time, in terms of their ability to provide quality service for the tasks and guarantee fairness amongst the jobs served. Furthermore, study the behavior of these scheduling algorithms and determine the most appropriate job scheduling algorithm for running jobs under cloud environment.

1.1. Review of Related Works

Job scheduling in cloud computing has attracted great attention. Most research in job scheduling adopt a paradigm in which a job in cloud computing system is characterized by its workload, dead-line and the corresponding utility obtained by its completion before deadline, which are factors considered in devising an effective scheduling algorithm. This paradigm is known as Utility Accrual (UA) paradigm.

Many researchers have proposed different scheduling algorithms that run under cloud computing environment. Most of the scheduling algorithms that have been proposed attempt to achieve two main objectives namely, to run the user task within the deadline and to maintain efficiency (load balancing) and fairness for all tasks (Li *et al*., 2010; Gupta and Rakesh, 2010; Yang *et al*., 2011). Here, we reviewed the most relevant research works done in the literature for job scheduling in cloud computing.

Garg *et al*. (2009) addressed the issue of increases in energy consumption by data centers in cloud computing. A mathematical model for energy efficiency based on various factors such as energy cost, CO2 emission rate, HPC workload and CPU power efficiency was proposed. In the model a near-optimal scheduling algorithm that utilizes heterogeneity across multiple data centers for a cloud provider was introduced.

Li *et al*. (2009) introduced a novel approach named EnaCloud, which enables application live placement dynamically with consideration of energy efficiency in a cloud platform. They use a VM to encapsulate the application, which supports the applications scheduling and live migration to minimize the number of running machines to save energy.

Furthermore, (Li *et al*., 2010) have addressed the problem of job execution in parallel processing in the cloud computing environment. To this end, they proposed a task scheduling mechanism using a pre emptive

mechanism that improves the utilization of resources in the clouds. Two feedback dynamic scheduling algorithms for this scheduling mechanism have been introduced to generate scheduling with the shortest average execution time of jobs.

The study in (Yang *et al*., 2011) highlighted the issue of job scheduling in cloud computing. They argued that there is no well-defined job scheduling algorithm for the cloud that considers the system state in the future. The existing job scheduling algorithms under utility computing paradigm do not take hardware/software failure and recovery in the cloud into account. To tackle this issue they proposed a Reinforcement Learning (RL) based algorithm that helps the scheduler in making scheduling decision with fault tolerable while maximizing utilities attained in the long term.

Li *et al*. (2011) introduced a hybrid energy-efficient scheduling algorithm using dynamic migration that handles job execution in private clouds. The algorithm concentrates on reducing the response time, con-serves more energy and performs higher level of load balancing.

In addition, the work in (Lin *et al*., 2011) concentrated on the issue of power consumption in data centers. They proposed a scheduling policy named Dynamic Round-Robin (DRR) that effectively reduces power consumption for virtual machine scheduling and consolidation. The algorithm attempts to deploy the virtual machines to servers and migrate virtual machines among servers.

The study in (Sindhu and Mukherjee, 2011) presented two scheduling algorithms for scheduling tasks in cloud computing, taking into account their computational complexity and the computing capacity of the processing elements. The algorithms are designed for private cloud environment where the resources are limited. The first algorithm is named Longest Cloudlet Fastest Processing Element (LCFPE) which considers the computational complexity of the cloudlets in the process of making scheduling decisions. The second algorithm is named Shortest Cloudlet Fastest Processing Element (SCFP). In this algorithm, the shorter cloudlets are mapped to Processing Elements (PEs) having high computational power so as to reduce flow time while at the same time taking into account that longer jobs are not starved. Lastly, (Paul and Sanyal, 2011) discussed the issue of how to utilize cloud computing resources proficiently and gain maximum profits with the job scheduling system. For this purpose, they proposed a credit based scheduling algorithm to evaluate the entire group of tasks in the task queue and find the minimal completion time of all tasks. The proposed scheduling method considers the scheduling problem as an assignment problem in mathematics where the cost matrix gives the cost of a task to be assigned to a re-source. However, the algorithm does not consider the processing time of a job, but other issues are considered such as the probability of a resource to be free soon after executing a task so that it will be available for the next waiting job.

2. MATERIALS AND METHODS

2.1. The Selected Job Scheduling Algorithms

In this study, four job scheduling policies in Cloud computing were carefully selected for evaluation, namely, Random, Round Robin (RR), Minimum Completion Time and Opportunistic Load Balancing. These algorithms are considered the most common and frequently used algorithms for job scheduling in Cloud computing. The aim of this study is to practically compare these algorithms. In the following we explain the details of each job scheduling algorithm.

2.2. Random Algorithm

The idea of random algorithm is to randomly assign the selected jobs to the available Virtual Machines (VM). The algorithm does not take into considerations the status of the VM, which will either be under heavy or low load. Hence, this may result in the selection of a VM under heavy load and the job requires a long waiting time before service is obtained. The complexity of this algorithm is quite low as it does not need any overhead or pre-processing. **Figure 2** demonstrates the process of assigning jobs to available VMs.

The detailed steps of random scheduling algorithm are illustrated in **Fig. 3**. The algorithm input includes two sets, namely cloudlets (i.e., jobs) and available VMs, including cloudlet list and VML. These two sets are measured by their sizes and are used by two variables calculated in steps 1 and 2 in the algorithm that are named Nocl and NoVM respectively. An index to the nominated VM is initialized to zero. The simulation process is done to handle the dynamic arrival of jobs. The index of the selected VM for the current job is computed randomly using Equation 1:

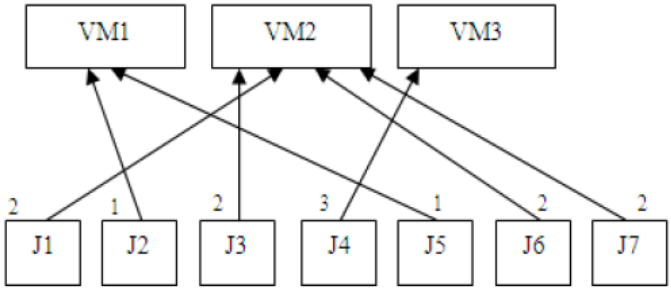

Fig. 2. The process of random algorithm

Input: *cloudletlist* : The list of cloudlets (i.e. jobs), *VML*: The list of available VMs
Output: Map each cloudlet to a VM

Steps
1 *Nocl* ← *cloudletlist.size*();
2 *NoVM* ← *VML.size*();
3 *index* ← 0;
4 for *j* ← 0 **to** *Nocl* **do**
5 *cl* ← *cloudletlist.get(j)*;
6 *index* ← *random*() × (*NoVM* − 1);
7 *v* ← *VML.get*(*index*);
8 *stagein* ← *TransferTime*(*cl, v, in*);
9 *stageout* ← *TransferTime*(*cl, v, out*);
10 *exec* ← *ExecuteTime*(*cl, v*);
11 if *(cl.AT + stagein + exec + stageout + v.RT ≤ cl.DL)* **then**
12 *sendjob*(*cl, v*);
13 *update*(*v*);
14 else
15 *Drop*(*cl*);
16 *FailedJobs*;
17 endif

Fig. 3. Random algorithm

Index = random() * (NoVM - 1) (1)

Where:
index = The index to the selected VM
random() = Function that returns a random value between 0 and 1
NoVM = The total number of available VMs

2.3. Round Rubin Algorithm

The Round Rubin (RR) job scheduling algorithm considered in this study distributes the selected job over the available VMs in a round order where each job is equally handled. The idea of the RR algorithm is that it attempts to sends the selected jobs to the available VMs in a round form.

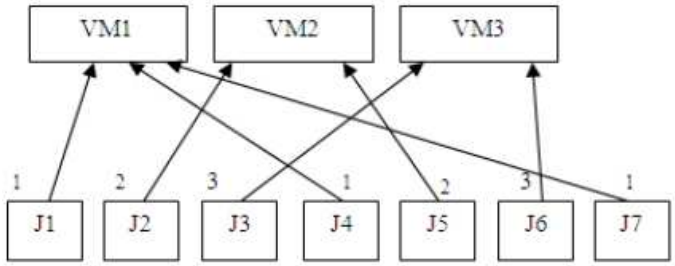

Fig. 4. The process of Round Robin algorithm

Input: *cloudletlist* : The list of cloudlets (i.e. jobs), *VML*: The list of available VMs
Output: Map each cloudlet to a VM

Steps
1 *Nocl* ← *cloudletlist.size*();
2 *NoVM* ← *VML.size*();
3 *index* ← 0;
4 for *j* ← 0 **to** *Nocl* **do**
5 *cl* ← *cloudletlist.get*(*j*);
6 *index* ← (*index*+1) mod *NoVM*;
7 *v* ← *VML.get*(*index*);
8 *stagein* ← *TransferTime*(*cl, v, in*);
9 *stageout* ← *TransferTime*(*cl, v, out*);
10 *exec* ← *ExecuteTime*(*cl, v*);
11 if *(cl.AT + stagein + exec + stageout + v.RT ≤ cl.DL)* **then**
12 *sendjob*(*cl, v*);
13 *update*(*v*);
14 else
15 *Drop*(*cl*);
16 *FailedJobs*;
17 end

Fig. 5. Round Rubin algorithm

Figure 4 depicts the mechanism of the Round Robin (RR) job scheduling algorithm. The algorithm does not require any preprocessing, overhead or scanning of the VMs to nominate the job's executor.

The detailed steps of Round Rubin job scheduling algorithm are illustrated in **Fig. 5**. The index of the selected VM for the current job is computed by a round robin fashion using Equation 2:

$$\text{index} \leftarrow (\text{index}+1) \bmod \text{NoVM} \quad (2)$$

Where:
index = The index to the selected VM

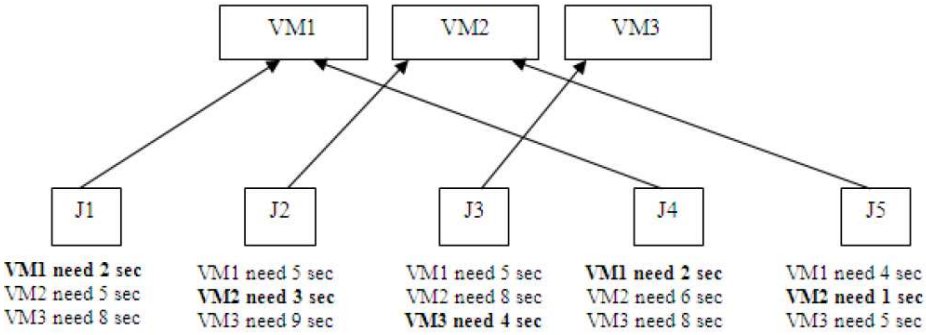

Fig. 6. The process of minimum completion time

Input: *cloudletlist* : The list of cloudlets (i.e. jobs), *VML*: The list of available VMs
Output: Map each cloudlet to a VM

```
1 initialization;
2 Nocl ← cloudletlist.size();
3 NoVM ← VML.size();
4 index ← 0;
5 for j ← 0 to Nocl do
6 cl ← cloudletlist.get(j);
7 min ← +∞;
8 for i ← 0 to NoVM do
9 v ← VML.get(i);
10 if min > (v.getready() + cl.getlength()/v.speed) then
11 min ← v.getready() + cl.getlength()/v.speed;
12 index ← i;
13 end
14 end
15 v ← VML.get(index);
16 stagein ← TransferTime(cl, v, in);
17 stageout ← TransferTime(cl, v, out);
18 exec ← ExecuteTime(cl, v);
19 if (cl.AT + stagein + exec + stageout + v.RT ≤ cl.DL) then
20 sendjob(cl, v);
21 update(v);
22 else
23 Drop(cl);
24 FailedJobs;
25 end
```

Fig. 7. The minimum completion time algorithm

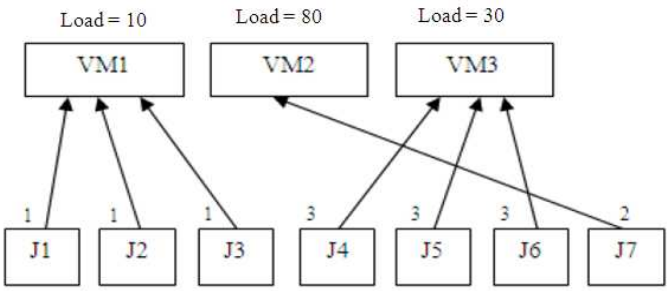

Fig. 8. The process of opportunistic load balancing

Input: *cloudletlist* : The list of cloudlets (i.e. jobs), *VML*: The list of available VMs
Output: Map each cloudlet to a VM

```
1 initialization;
2 Nocl ← cloudletlist.size();
3 NoVM ← VML.size();
4 index ← 0;
5 for j ← 0 to Nocl do
6 cl ← cloudletlist.get(j);
7 min ← +∞;
8 for i ← 0 to NoVM do
9 v ← VML.get(i);
10 if min > (v.getready()) then
11 min ← v.getready();
12 index ← i;
13 end
14 end
15 v ← VML.get(index);
16 stagein ← TransferTime(cl, v, in);
17 stageout ← TransferTime(cl, v, out);
18 exec ← ExecuteTime(c, v);
19 if (cl.AT + stagein + exec + stageout + v.RT ≤ cl.DL) then
20 sendjob(cl, v);
21 update(v);
22 else
23 Drop(cl);
24 FailedJobs;
25 end
```

Fig. 9. The opportunistic load balancing algorithm

NoVM = The total number of available VMs

2.4. Minimum Completion Time Algorithm

The Minimum Completion Time job scheduling algorithm attempts to allocate the selected job to the avail-able VM that can offer the minimum completion time taking into account its current load. The main criterion to determine the VM in the minimum completion time scheduling algorithm is the processor speed and the cur-rent load on each VM. The algorithm first scans the available VMs in order to determine the most appropriate machine to perform the job. Subsequently, it dispatches the job to the most suitable VM and starts execution. **Figure 6** illustrates the process of job scheduling using the minimum completion time algorithm. Notice that all the available VMs (VM1, VM2 and VM3) are able to run the set of jobs but with different response time. For that reason, job J1 is send to VM1 as it is the fastest machine that can run the job and return the results within a short time, which is 2 sec. VM2 and VM3 can also run J1 but with longer time consumption, namely 5 sec for VM2 and 8 sec for VM3.

The detailed steps of the minimum completion time scheduling algorithm are presented in **Fig. 7** and are repeated for each job. Formally, the index for the selected VM that will execute the current cloudlet cl is computed using formula (3) Equation 3:

$$\text{index} \leftarrow \text{Min}\{\text{v.getready()}+\text{cl.length/v.speed}|\forall v \in \text{VML}\} \quad (3)$$

2.5. Opportunistic Load Balancing Algorithm

This algorithm attempts to dispatch the selected job to the available VMs which has the lowest load compared to the other VMs. The idea is to scale the current loads for each VM before sending the job. Then, the VM that has the minimum load is selected to run the job. **Figure 8** illustrates how the opportunistic load balancing algorithm works. Assume that there are three virtual machines (VM1, VM2 and VM3) with different loads, namely 10 sec for VM1, 80 sec for VM2 and 30 sec for VM3. Let Ji be a new job that has been arrived for execution. The scheduler should select one of the VMs to run Ji. The scheduler will make a decision by selecting the VM1 to run the new job Ji as it has the minimum load, which is 10 sec. The meaning of load here is indicated by the level of VM preoccupation with current jobs. In other words, VM1 will finish the as-signed jobs after 10 sec; VM2 will finish the as-signed jobs after 80 sec and VM3 will finish the assigned jobs after 30 sec. For that reason the scheduler selects VM1, which has the lowest load. Basically, the opportunistic scheduling algorithm is considered one of the best choices in load balancing.

Mathematically speaking, the index to the selected VM that will execute the current cloudlet cl is calculated using (4) Equation 4:

$$\text{index} \leftarrow \text{Min}\{\text{v.getready()} \mid \forall v \in \text{VML}\} \quad (4)$$

Figure 9 presents the detailed step of the opportunistic load balancing job scheduling algorithm that is repeated for each job.

3. RESULTS AND DISCUSSION

The performances of the developed algorithms are evaluated under different evaluation criteria. All algorithms are implemented and tested using in Cloudsim simulator. The implementation has been accomplished by modifying the original source code of the simulator that was written in the Java language. Net beans 7.1, a Java editor was used for this purpose.

3.1. Performance Metrics

Various performance metrics were taken into consideration in order to measure and evaluate the selected job scheduling algorithms. These metrics include the make span, amount of throughput and total cost. These performance metrics are the most important and frequently used metrics in the previous works for evaluating the scheduling algorithms in cloud computing environment. These performance metrics are further explained below.

3.2. Makespan

The makespan represent the maximum finishing time among all received jobs per time. This parameter shows the quality of job assignment to resources from the execution time perspective. The formal formula for the Makespan is shown in Equation 5:

$$\text{Makespan} = \text{Max}\{FT_j \mid \forall j \in J\} \quad (5)$$

Where:

FTj = The finishing time of job j belonging to the job list J

j = Job from the list of jobs

J = List of jobs

3.3. Throughput

In this study, each job is assumed to have hard dead-lines which represent the finishing time. Therefore, the throughput is the number of executed jobs, which is calculated to study its efficiency in satisfying the jobs dead-lines. The throughput is calculated using Equation 6:

$$\text{Throughput} \quad J = \sum_{j \in J} X_j \tag{6}$$

Where Xj is:

$$X_j = \begin{cases} 1, & \text{job j has finished execution} \\ 0, & \text{Otherwise} \end{cases}$$

Where:
j = Job from the list of jobs
J = List of jobs

3.4. Total Cost

If the basic concept of Could computing is renting resources for customers, then the total cost needed for executing the list of jobs is essential for evaluating system performance. The total cost is calculated based on the processing time and the amount of data transferred. Equation 7 illustrates how the total cost is computed:

$$\text{Total Cost (TC)} = P_j * PC + \left(\sum_{f \in Fin_j} \text{Size(f)} + \sum_{f \in Fout_j} \text{Size(f)} \right) \times TrC \tag{7}$$

Where:
f = Single file
TC = The total cost
Pj = The processing time of the job j
PC = The processing cost
TrC = The cost of transferring the input files (Finj) and the output files (Fout j)

3.5. Dataset

The Ligo real dataset (Brown *et al*., 2007) is used, which are The Laser Interferometer Gravitational Wave Observatory (LIGO) attempts to detect gravitational waves produced by various events in the universe as per Einstein's theory of general relativity. The dataset was represented by a set of XML files that vary in the number of jobs. Each file contains a set of jobs and their requirements such as job length, job ID, lists of input and output files and their sizes. Set of six files has been taken to be the workload of this research where the number of jobs is 50, 100, 200, …, 1000. The missing parameters in job's characteristics such as the arrival time, file location and the start and finish deadlines are completed randomly based on the job's length and data size found in the XML files.

3.6. Simulation Results

Three experiments were carried out in this study. All experiments aim at analyzing several performance metrics (throughput, makespan and total cost) using the cloudsim simulation tool with respect to the various number of jobs. For each of the experiments, various numbers of scenarios with different parameters values were taken into consideration during simulation. **Table 1** summarizes the simulation parameters used in the experiments.

3.7. The Experiment Results of the Throughput Percentage

Figure 10 depicts the result of the throughput percent-age for the jobs given for each scheduling algorithm. From the figures, it can be concluded that the throughput deteriorates for all cases when the number of jobs were increased for all scheduling algorithms. This is mainly due to the increasing number of jobs, resulting in a high load for each VM that further leads to the failure to exe-cute some jobs.

From the figure, it can also be noted that the minimum completion time steadily outperforms the other scheduling algorithms in all cases. This is because the minimum completion time assigns the job to the most appropriate VM that is able to accomplish the job within the given constraints. The throughput for the minimum completion time reached up to 100% when the number of jobs was at 50. The throughput was reduced dramatically when the assigned jobs increased and reached 30% when the number of jobs reached 1000 jobs.

Table 1. The parameters setting

Parameter	Value
Number of VMs	100
Number of Jobs	50-1000
Transmission cost	0.10 USD
Processing cost	0.10 USD

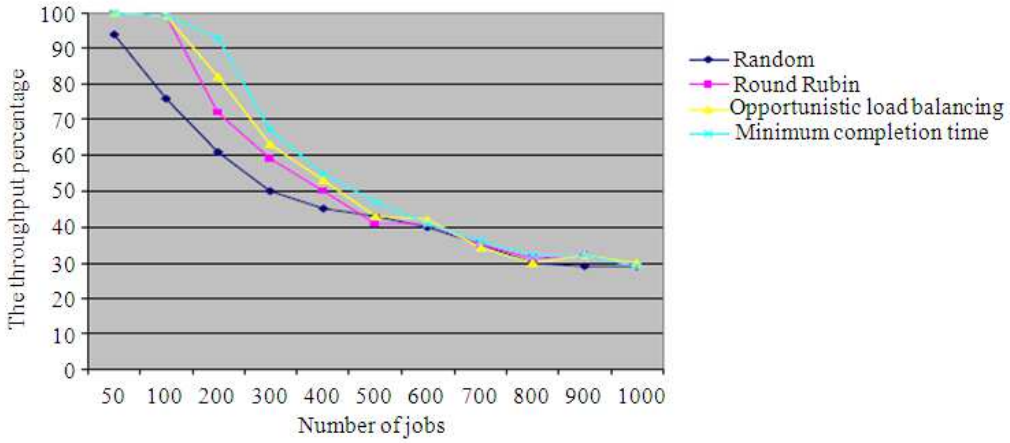

Fig. 10. Throughput percentage

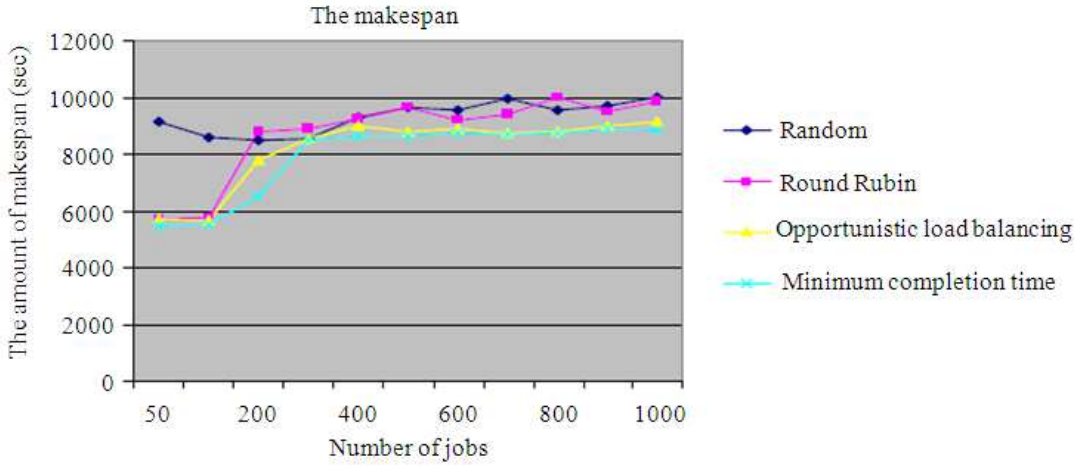

Fig. 11. The makespan

The opportunistic load balancing algorithm performed better than the Round Rubin and the random algorithms in most of the cases. This is because the opportunistic load balancing algorithm attempts to distribute the jobs to the avail-able VM which had the lowest load compared to the other VMs. Notice also that the algorithm performance deteriorated when the number of jobs increased. The throughput reached up to 100% when the number of jobs was less than 100, while the throughput decreased significantly when the number of jobs reached 1000.

However, the Round Robin algorithm performance is low because it does not take into account the specific VM‘s load and handled the jobs in sequence by giving the same time portion for each job. Finally, the Random algorithm performed the worst in most of the cases compared to the other scheduling algorithms. This is because the random algorithm randomly assigns the selected jobs to the available (VM). The algorithm does not take into considerations the VM status whether it was under high or low load.

3.8. The Experiment Results of the Makespan

This experiment focuses on the quality of job assignment to resources from the perspective of the execution time. **Figure 11** illustrates the observation of the makespan time with increasing numbers of assigned jobs for each scheduling algorithm.

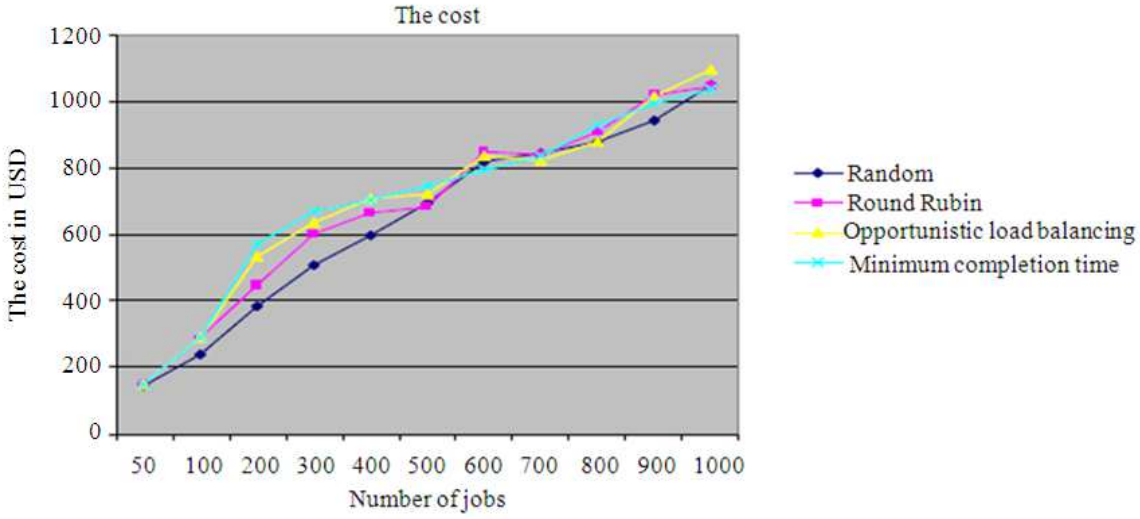

Fig. 12. The total cost

From the figure it is clear that the minimum completion time has achieved the lowest value of makespan in all cases compared to the other algorithms. This is mainly due to the fact that the mini-mum completion time attempts to select the most suitable VM that can rapidly respond and execute the given job and generate the output to the user. Notice that for the minimum completion time, the makespan time is increased when the number of jobs increases. The total makespan time that is required to run 50 jobs is almost 5500 seconds, while the algorithm required 8800 seconds as a makespan time when the number of jobs reach to 1000 jobs.

The opportunistic load balancing algorithm achieved better compared to the Round Rubin and Random algorithms. Notice that the opportunistic load balancing achieves the same with the minimum completion time when the number of jobs is less than 200. This is because the two algorithms have almost similar criteria in deter-mining the most appropriate VM to run the job. On the other hand, the Round Rubin algorithm performed the worse compared to the minimum completion time and the opportunistic load balancing algorithms in all cases. This is because the Round Rubin algorithm is not concerned with the given VM specifications and loads the job in a circulatory form. The Random algorithm is the worst among the other algorithms for achieving the highest makespan time. This is because the random algorithm attempts to randomly distribute the set of jobs over the VM and the job constraints is not taken into consideration.

3.9. The Experiment Result of the Total Cost

In this experiment we aim to study the impact of the number of jobs on the total cost when VMs execute their assigned jobs. **Figure 12** illustrates the experimental results obtained for the total cost consumed by each set of jobs fed to the four scheduling algorithms. It is clear that the total cost is highly influenced by the number of assigned jobs for every scheduling algorithm. Notice that the minimum completion time produces the highest cost in all cases compared to the other scheduling algorithms. This is mainly due to the minimum completion time accomplishing the largest number of received jobs. Thus, the total cost will be more compared with the other algorithms. The opportunistic load balancing scheduling algorithm incurred higher cost compared with the Random and Round Rubin algorithms. This is because the opportunistic load balancing has the capability to run more jobs at the same time, as the algorithm when the jobs are dispatched over the available VM while taking the VM load into account. Thus, many jobs can be run and that will lead to an increase of cost. The Round Rubin algorithm produced less cost compared to the minimum completion time and the opportunistic load balancing algorithms. The Random algorithm is the superior in all cases in terms of the total cost compared with the other algorithms. Nevertheless, the Random algorithm has the same cost with the Round Rubin algorithm when the number of jobs is up to 500, 600 and 700. Moreover, the Random algorithm possesses the same cost with the opportunistic load balancing

algorithm when the number of jobs is in range of 700-800. From this experiment we can conclude that cost has a strong relationship with the number of executed job and the total amount of the cost highly dependent of the executed jobs.

4. CONCLUSION

In this study, the behavior of four job scheduling algorithms, namely: Random, Round-Rubin (RR), Opportunistic Load Balancing and Minimum Completion Time have been investigated and examined in a Cloud computing environment. These job scheduling policies have been extensively evaluated by focusing on the major characteristics of the cloud computing environment. Based on the simulation results, it is shown that some of the scheduling algorithms are beneficial to be used in Cloud computing. Based on the results, it can be also concluded that there is not a single scheduling algorithm that provides superior performance with respect to various types of quality services. This is because job scheduling algorithms needs to be selected based on its ability to ensure good quality of services with reasonable cost and maintain fairness by fairly distribute the available resources among all the jobs and respond to the constraints of the users.

5. REFERENCES

Brown, D., P. Brady, A. Dietz, J. Cao and B. Johnson *et al*., 2007. A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis. Workflows E-Sci. DOI: 10.1007/978-1-84628-757-2_4

Garg, S.K., C.S. Yeo, A. Anandasivam and R. Buyya, 2009. Energy-efficient scheduling of HPC applications in cloud computing environments. Comput. Sci. Distributed, Parallel Cluster Computing.

Gupta, P.K. and N. Rakesh, 2010. Different job scheduling methodologies for web application and web server in a cloud computing environment. Proceedings of the 3rd International Conference on Emerging Trends in Engineering and Technology, Nov. 19-21, IEEE Xplore Press, Goa, pp: 569-572. DOI: 10.1109/ICETET.2010.24

Li, B., J. Li, J. Huai, T. Wo and Q. Li *et al*., 2009. EnaCloud: An energy-saving application live placement approach for cloud computing environments. Proceedings of the International Conference on Cloud Computing, Sept. 21-25, IEEE Xplore Press, Bangalore, pp: 17-24. DOI: 10.1109/CLOUD.2009.72

Li, J., J. Peng and W. Zhang, 2011. A scheduling algorithm for private clouds. J. Convergence Inform. Technol., 6: 1-9.

Li, J., M. Qiu, J. Niu, W. Gao and Z. Zong *et al*., 2010. Feedback dynamic algorithms for preemptable job scheduling in cloud systems. Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology, Aug. 31-Sep. 3, IEEE Xplore Press, Toronto, ON, pp: 561-564. DOI: 10.1109/WI-IAT.2010.30

Lin, C.C., P. Liu and J.J. Wu, 2011. Energy-aware virtual machine dynamic provision and scheduling for cloud computing. Proceedings of the 4th International Conference on Cloud Computing, Jul. 4-9, IEEE Xplore Press, Washington, DC., pp: 736-737. DOI: 10.1109/CLOUD.2011.94

Paul, M. and G. Sanyal, 2011. Task-scheduling in cloud computing using credit based assignment problem. Int. J. Comput. Sci. Eng., 3: 3426-3430.

Sindhu, S. and S. Mukherjee, 2011. Efficient task scheduling algorithms for cloud computing environment. Commun. Comput. Inform. Sci., 169: 79-83. DOI: 10.1007/978-3-642-22577-2_11

Yang, B., X. Xu, F. Tan and D.H. Park, 2011. An utility-based job scheduling algorithm for cloud computing considering reliability factor. Proceedings of the 2011 International Conference on Cloud and Service Computing, Dec. 12-14, IEEE Xplore Press, Hong Kong, pp: 95-102. DOI: 10.1109/CSC.2011.6138559