A Best Effort Heuristic Algorithm for Scheduling Timely Constrained Tasks in the Cloud

Riyadh I. Louis, Ahmed I. Saleh, Mohammed F. AL Rahmawy
Hazem M. El-Bakry , Samir M. Abd El-razek

Abstract— The size and complexity of Cloud systems are growing more rapidly, and hence, the management of these cloud systems and its resources is a major research area. Resource provision with respect to SLA (Service Level Agreement) is directly tied up with customer satisfaction like providing the service with less Cost with less finshing time, for that, cost effective scheduling with real time constraints are major challenges in adopting cloud computation. In this work we propose a two-stage scheduling technique for timely constrained cloud computing services. The first stage is in charge of producing a scheduling sequence, whereas the second stage aims to dispatch tasks to computing nodes of a cloud computing system. The two stages are independent of one another and; therefore, one can change a policy in one stage without configuring another one. The main goal of the proposed work is to improve user satisfaction, to balance the load efficiently and to bolster the resource utilization and provide the service with the competitive cost at the same time.

Index Terms— Cloud Computing, Real time, Scheduling, Resource Utilization.

1 Introduction

Nowadays cloud environment is used in most of the business organizations and educational institutions.The ultimate definition for the cloud computing has been developed by NIST was the cloud computing is a model for allowing convenient, on demand network access to a shared huge number of configurable resources such as networks, servers, storage, applications, and services that can be quickly provisioned and released with minimal management effort or service supplier communication.Cloud computing and service-oriented architectures are driving a shift toward distributed real-time systems, where the success of applications dosent depends on only on the correctness of result but also quality-of-service (QoS) performance (i.e., deadlines and availability) [1].A service level agreement (SLA) is applied as a contract between customers and service providers to accomplish this. In this study, we focus on scheduling techniques that allocate computing resources to tasks in a way to satisfy deadline requirements to fulfill a specified SLA in distributed systems (e.g., Hadoop computing environments) [2].

In cloud computing environments, there are two important players: *cloud providers* and *cloud users*. Providers hold enormous computing resources in their large datacenters and rent resources out to users on a per-usage basis. On the other hand, there are users who have applications with fluctuating loads and hire resources from providers to run their applications. In most cases, the interaction between providers and users occur as shown in Figure 1 [3],as we can see the user demands are low cost with minimum finshed time for there application,while from the cloud provider point of view he want finsh the applications with maximum revenue by maximized the resources utilization.

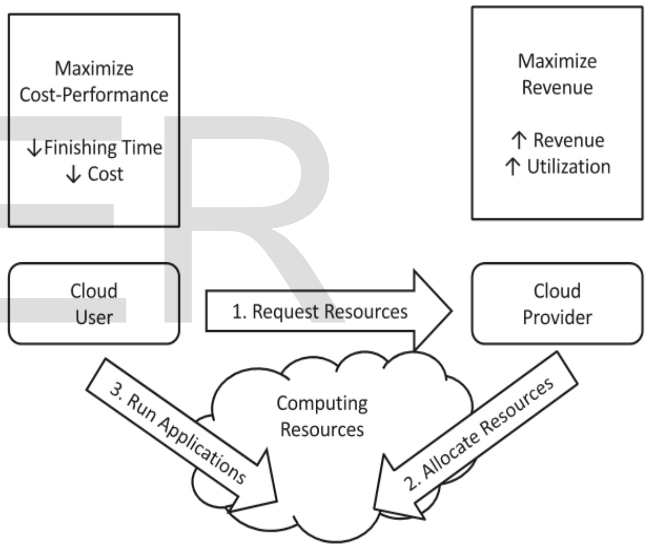

Fig 1: Cloud Usage Scenario

The cloud as a distributed system that consists of loosely coupled computing nodes connected using a computer networking to achieve high performance Scheduling schemes play an important role in distributed systems [4]. Scheduling algorithms are divided into two categories, namely, dynamic scheduling [5] and static scheduling [6]. To satisfy a given SLA of timely constrained applications in the cloud, scheduling mechanisms are responsible for ensuring that all tasks complete before their deadline [7].In cloud computing, large numbers of users submit their tasks to the cloud broker which transfers the request to the Cloud Service Providers (CSP) as explained by Kwang Mong Sim [8]. CSP provides the services transparently to the users independent of host infrastructure through virtualization. Virtualization is a technique that logically separates the physical resource. Each logical unit of physical resource acts as a VM. The necessity of virtualization is to provide hardware independency, software isolation, reduction of energy consumption and security with increased resource utilization [9].The tasks, if not properly scheduled in the cloud environment, may lead to network congestion. Therefore, more numbers of tasks are discarded due to network congestion. A good scheduling algorithm should speed up the task execution to reduce network traffic. Consequently, the user satisfaction and the number of tasks accepted for execution increases that will boost the revenue of the CSP and also reduce the local network traffic.

Timely constrained applications services are required to satisfy a dual notion of correctness: not only must the correct value be determined, this value must be determined at the correct time. In hard real-time systems, certain pieces of computation have deadlines associated with them, and it is imperative for the correctness of the system that all such pieces of computation complete by their deadlines. (In contrast, soft and firm real-time systems may allow for an occasional deadline to be missed, or for a deadline to be missed by no more than a certain amount, etc.) This paper focuses almost exclusively upon the scheduling of soft real-time systems in the cloud.

Once the resources are provisioned to the submitted applications (cloud), each application needs to schedule at the allocated resources to perform various computation tasks. In this context, the scheduling problem concerns matching the tasks to the available resources for maximization of system throughput, execution efficiency, and so on. The optimal matching is an optimization problem with NP-complete complexity. Due to the high diversity of tasks and situations, there is no general task scheduling algorithm that can fit for all tasks.
Classifying task scheduling methods into static scheduling and dynamic scheduling. Static scheduling techniques are suitable for the environments where the details of all tasks and resources are known prior to the scheduling being performed. On the contrary, dynamic task scheduling is performed on the fly each time a task arrives. Dynamic scheduling techniques are applied in the environments where task information and resource states cannot be available in advance.

A computing system needs to run on a specific computer processing platform. The platform may be a uniprocessor, consisting of one processor or multiprocessor consisting of several processors. the cloud systems use multiprocessors,where the individual processors may all be the same or they may differ from one another. Multiprocessors environment can be divide into three different categories based on the speeds of the individual processors.

• *Unrelated heterogeneous multiprocessors.* In these platforms, the processing speed depends not only on the processor, but also on the task being executed. For example, if one of the processors is a graphics coprocessor, graphics tasks would execute at a more accelerated rate than non-graphics tasks. Each (processor, task)-pair of an unrelated heterogeneous system has an associated speed $s_{i,j}$ which is the amount of work completed when task j executes on processor i for one unit of time.

• *Uniform heterogeneous multiprocessors.* In these platforms, the processing speed depends only on the processor. Specifically, for each processor i and for all pairs of tasks j and k, we have $s_{i,j} = s_{i,k}$. In these multiprocessors, we use a si to denote the speed of the i'th processor.

• *Identical multiprocessors.* In these platforms, all processing speeds are the same. In these systems, the speed is usually normalized to one unit of work per unit of time [10].

There are two fundamental classes of multiprocessor schedulers: *global* and *partitioned*. Under global scheduling (illustrated in inset (a) of Figure 1), all processors serve a single ready queue and tasks may migrate among processors. In contrast, under partitioned scheduling (illustrated in inset (b) of Figure 2), tasks are statically assigned to processors during an offline phase and each processor is scheduled individually using a uniprocessor policy [11].

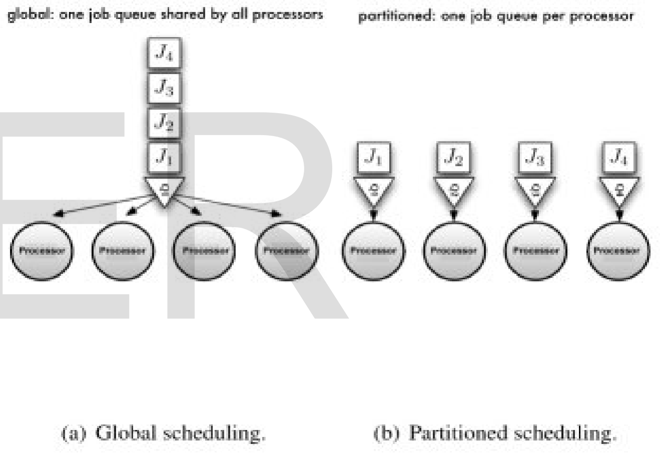
global: one job queue shared by all processors
partitioned: one job queue per processor
J_4
J_3
J_2
J_1
J_1
J_2
J_3
J_4
Processor
(a) Global scheduling.
(b) Partitioned scheduling.

Figure 2: Illustration of multiprocessor scheduling approaches

The remainder of this paper is organized as follows: section II briefly reviews some related works. Section III comes up with the system model. Section IV describe the proposed algorithm while section V has the experiment example and results. Section VI ends with some conclusions and future works.

2 Related Work

This section reviews various scheduling algorithms developed to schedule the tasks depending on the type of task or resource in a cloud environment.
The first element in tasks scheduling is the task it self and it classified as ***batch tasks***, ***transactional tasks*** and ***interactive tasks*** based on their characteristics as explained by Y. Zhang, et al. and D. Carrera, et al. [13, 14].
This research spotlights on batch task scheduling and hence the literatures have been restricted to batch task scheduling.

The batch task scheduling is categorized into two types as static and dynamic scheduling depending upon the characteristics of scheduling. In static scheduling, the tasks, which are executed in certain resources, are also non-preemptive. Unlike static scheduling, the tasks are scheduled at the run time in dynamic scheduling that supports migration and preemption.The tasks are generally classified into *deadline based* and *non-deadline based* tasks depending upon the user input.

EDF Algorithm Deadline based tasks are scheduled using Earliest Deadline First (**EDF**) algorithm to complete earliest deadline tasks within their deadline as developed by V. Gamini Abhaya, et al. [15]. **EDF** is a type of priority scheduling. The tasks are prioritized based on not only deadline, but also the arrival time, waiting time and so on. The tasks are dynamically prioritized and mapped to the VMs with limited support of migration in dynamic scheduling. The task preemption and task migration can fritter away execution time and network bandwidth as developed by M. Stillwell, et al. [16].

Min-Min Algorithm The scheduling objective in Min-Min is to achieve Minimum Completion Time. The scheduling process is done by adding all tasks to a set known as the meta task, if the meta task not empty, the algorithm begins to calculate the completion time for each task; then, the task that has the earliest minimum execution time is taken from the set and assigned to the corresponding resource. Then, this task is removed from the metatask set. This process repeats after removing this task till all tasks in meta-task are processed [17].

Max-Min Algorithm This algorithm works in a way unlike (Min-Min)algorithm method, where it choose the task which has the maximum execution time and assign it to the resource has the minimum completion time [18]. Max-Min is better than Min-Min algorithm in resource utilization [19].

Activity Based Costing in Cloud Computing (ABC Algorithm) This algorithm calculate the cost of the resource and applies the concept of cost-based priority by calculating the cost of each individual use of the resources and the profit of using these resources. According to the calculations, it gives tasks priorities and sorted in three levels; High, Medium and Low level priority, where the tasks with highest profit have the highest priority. If new task arrives its priority calculated and it is assigned to the end of the appropriate level [20].

3 The Proposed Algorithm

3.1 Characteristic of tasks

In the proposed algorithm the incoming tasks are assumed as ***batch*** of tasks. Each tasks is ***aperiodic*** (i.e. the arrival time of the task is not known in advance) and ***independent*** of each other (i.e. the input of one task does not depend on the output of other tasks) as modeled by Chenhong Zhao, et al. [21].

The incoming tasks are assumed as ***non-preemptive*** (i.e. even if a high priority task arrives, the task in execution is not preempted). The tasks (***T***) are defined as $T = \{t_1, t_2 \dots t_n\}$, where ***n*** represents the number of tasks

It is assumed that the user must specify the length (***l***) and the corresponding deadline (***d***) of the task during submission

$$T_i = \{l_i, d_i\}; \quad i \in (1, n).$$

The length of the task or the size of the task is expressed as the number of instructions required for processing the task. It is generally defined as number of Million Instructions (**MI**) required for processing the submitted task [22]. The tasks may request either computational resources or storage resources. In this work, it is assumed that the task request only computational resources for their execution.

3.2 Characteristics of Resources

The resources (i.e. VMs) are independent of each other. VMs may exist either in ***homogeneous*** or in ***heterogeneous*** multiprocessor environments. The processing speed of the VMs in a ***homogeneous*** environment is defined as

$$S_1 = S_2 = \dots = S_n$$

Here, all VMs have equal processing speed.

The processing speed of the VMs in a ***heterogeneous*** environment is defined as

$$S_1 \neq S_2 \neq \dots \neq S_n$$

So that all VMs have different processing capacity. We assume the maximum speed ($\mathbf{Max_s}$) is the highest processing speed in resource pool.where $\mathbf{Max_s = Max\,(S_1, S_2, \dots, S_n)}$

3.3 Algorithm Policy

In the above scenario, a large number of users submit their tasks in the cloud. Among them, some may request more processing speed than the available processing speed of the VM that may affect the subsequent tasks. These tasks are filtered to reduce the number of tasks violating their deadline and also to increase the performance of the system or to find another VM with higher speed capable of satisfying the deadline. Every task contains two attributes namely length and deadline during its submission. The scheduler can effectively schedule and complete the tasks within their deadline by ***prioritizing*** the tasks. The ***priority*** scheduler can calculate the priority value based on different parameters like ***waiting time*** of the task, ***length*** of the task and ***deadline*** of the submitted tasks. It does not focus on resource utilization and previous workload. The priority scheduler can efficiently schedule the tasks to the underlying VM so that it can reduce the ***waiting time*** (***Wt***) of the tasks. It may also increase the ***throughput*** (**Tp**) of the system. The relationship between ***waiting time***, ***throughput*** and ***resource utilization*** (***Ru***) are described below:

$$Tp \;\alpha\; Ru$$

And

$$Tp \;\alpha\; 1/\,Wt$$

This work proposes an Adaptive Two-Stage Scheduling System, see figure 3 that filters, prioritizes and maps the task to a suitable VM. The first stage is in charge of producing a scheduling sequence, whereas the second stage aims to dispatch feasible tasks to computing nodes of a distributed system. The proposed work aims to improve system performance and resource utilization.

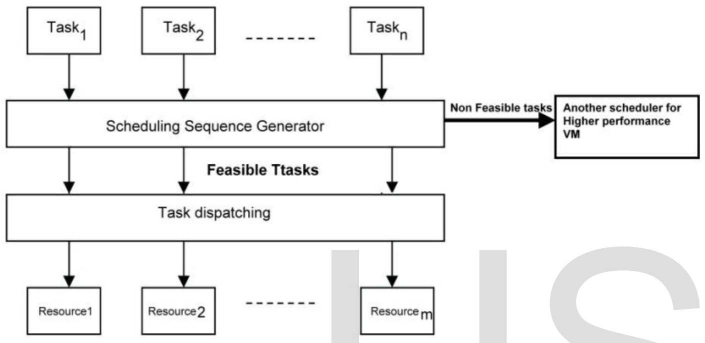

Fig 3: Decomposing an entire scheduling mechanism into two distinct Stages.

4. Design of Adaptive Two-Stage Scheduling Technique

The tasks are submitted from various users with different demand and the tasks are assumed to need only computing resources for their execution. The submitted tasks are congregated to the Cloud User Interface (i.e. Portal). The minimum processing speed required by the task is dynamically estimated depending on the user input. Sometimes, the task may require more processing speed than the available processing speed of the resource. In this work, the tasks are accepted only if it can adapt and complete within deadline using the available processing speed of the VM and eliminate the task that require more processing speed than the available processing speed of the resource.
The algorithm comprises of two components for scheduling the tasks in multilevel manner.In the first stage, preprocessing and filtering are done to exlude the tasks that require more processing speed than the available processing speed of the VM. These tasks are handed over to the deadline reassignment, while the accepted tasks assigns priority dynamically. The prioritized tasks are then passed to the next level of the algorithm. In the second level, tasks are maped dynamically to suitable resources to complete its execution within deadline and normal cost. Moreover, the algorithm also balances the system load. After resource allocation.

4.1 Preprocessing tasks

The VM exists either in ***homogeneous*** or ***heterogeneous*** environment depending on the data center policy. And there are two functions such as either accepted or rejected for tasks are carried out, the tasks processed are considered as a Bernoulli distribution (i.e. Acceptance is treated as success and rejection is treated as a failure) as each task has two possible outcomes and independent of each other. The minimum processing speed or computation speed required for the submitted task can be represented as S_{min} and can be calculated as shown in equation (1)

$$\mathbf{S_{min}} = \frac{l_t}{d_t} \quad \text{……….. (1)}$$

l_t And d_t stand for the length and deadline of the task.

The processing speed of the VM is expressed in **MIPS** (Million Instructions per Second).The maximum processing speed of existing VM is represented as $\mathbf{Max_s}$.

Homogeneous Envirnment

$S_1=S_2=\ldots=S_n$ speed of homogeneous environment

Max speed $\mathbf{Max_s}= s_1=s_2=\ldots= s_n$ and

In **Heterogeneous Envirnment**

$S_1 \neq S_2 \neq \ldots \neq S_n$ speed of heterogeneous environment

Max speed $\mathbf{Max_s}= \max(s)$

The tasks are feasible by comparing S_{min} and $\mathbf{Max_s}$ only if $\mathbf{S_{min} \leq Max_s}$

We put feasible tasks in queue **Q1**. Otherwise, the tasks are rejected and passed to deadline reassignment which rescheudling tasks on other VM(s) or even other cloud.
The total processing speed of the reserved VM(s) is denoted as S_t and is calculated as in equation (2).

$$\mathbf{St}=\sum_{i=1}^{n} S_i \quad \text{.....(2)}$$; where n=no. of processors in the reserved VM(s).

The tasks are preprocessed based on the minimum**Time Required for Processing** already accepted tasks (**TRP**) in a queue **Q1** and it is computed as given in eqution (3).

$$\mathbf{TRP}=\frac{\sum_{t=1}^{n1} l_t}{s_t} \quad \text{.....(3);}$$ where **n1** no. of tasks in queue **Q1**

Then we calculate the average time required for processing single task **AVG** as in eqution (4).

$$\mathbf{AVG=TRP/n1} \quad ..... (4)$$

We compare **AVG** and the deadline of task and If

$$\mathbf{AVG} \le d_t$$

Then we put the task in new queue **Q2.** Otherwise the task is rejected and and passed to deadline reassignment.

The Utilization of the data center

The utilization can be calculated as in eqution (5).

$$\mathbf{U}= \frac{\sum_{i=1}^{n2} \frac{l_i}{d_i}}{S_t * R} \quad .....(5);$$ where **n2** no. of tasks in queue **Q2**

R no. of resource

If $\mathbf{U \le 1}$ then the tasks in **Q2** can be processed otherwise the tasks are forwarded to another data center.

4.2 Priority Assignment

The priority value of the tasks stored in the **Q2** priority queue is calculated based on different parameters like length of the submitted task, deadline of the task, waiting time of already
accepted task and the maximum computational speed of VM. The optimistic average computation time for processing the current task is represented as T_{Ct}. It is computed based on different parameters like length of the task and processing speed of the VM as shown in equation (6).

$$\mathbf{T_{Ct}} = \frac{l_t + l_{t-1}}{Max_s} \quad .... (6)$$

Then we calculate the priority (**Pt**) as in eqution (7).

$$\mathbf{Pt} = \left(\frac{\left(\frac{l_t}{d_t - \mathrm{TCt}}\right)}{Max_s} \right)^{-1} \quad ...... (7)$$

The VM with maximum processing speed is taken into the account instead of checking with every VM because, if the VM with maximum processing speed cannot complete the task within deadline, then no other VM is capable to complete the tasks within deadline. The tasks are sorted and stored in a queue (**Q**$_{\text{final}}$) based on their priority value. The task with the lowest priority value is given the highest preference and it remains in the head of the queue.The prioritized tasks are passed to the second stage. The overall first stage of the algorithm can be represent by the pseudo code below:

```
Step1: for all resource in resource pool DO
        Calculate $Max_s$
        Calculate $S_t$ from equation (2)
    End for
Step2: for all tasks in batch queue Do
        Calculate $S_{min}$ from equation (1)
        Compare $S_{min}$ and $Max_s$
            If $Smin$ less than $Max_s$
                Put task in queue Q1
            Else
                Forword task to deadline reassignment
            End if
    End for
Step3: for all tasks in queue Q1 DO
        Calculate TRP from equation (3)
        Calculate AVG from equation (4)
        Compare AVG and $d_t$
            If AVG less than $d_t$
                Put task in queue Q2
            Else
                Forword task to deadline reassignment
            End if
    End for
Step4: for all tasks in Queue Q2 DO
        Calculate $T_{Ct}$ from equation (6)
        Calculate priority $P_t$ from equation (7)
    End for
Step5: put all tasks in Q2 in $Q_{final}$ with its priority
Step6: Sort all tasks in $Q_{final}$ by priority in Ascending
```

4.3 Dispatching tasks (Stage Two)

Resource management includes scheduling of tasks and resource reutilization in order to complete all tasks on time. As users demands can increase at any time in a cloud environment so there will be a need for a smart and efficient algorithm in order to manage all the resources so that customer's requirements can be fulfilled. Profit can increase only by managing the resources from a provider's point of view. The user will take service from the provider who will charge him least among all providers for the same service.

Most of the traditional scheduling algorithms in cloud computing don't make any consideration for the task's cost, where the task is assigned to any available resource as soon as it arrives. This makes small problems such as "over-costed" and/or "over-priced" cloud services in case of high volume simple tasks and "under-costed" and/or "under-priced" in low volume complex ones. To beat these problems we proposed algorithm aims not only to the minimization of the services completion time and maximizing the resource utilization , in order to enable the provider to provide the best and most efficient services with accepted competitive prices.

To calculate the cost of tasks we use eqution (8).

Cost= $\sum_{1}^{j} R_j$ *** cost of** R_j **… (8);** where j is no. of resources

Once the task's priority is calculated, the task is sent to the appropriate resource in the scheduler, where the algorithm, dispatced the task(s) which has/have the highest calculated priority and longest length, to the resource(s) which has the minimum completion time to make the resource(s) run in smallest time to a void missing the deadline of the task(s) and to reduce the cost as explained in pseudo code next in details:

```
Step 1: For all available resources in resources pool DO
        Sort resources by speed in Descending
        End for
Step 2: For all tasks in queue $Q_{final}$ DO
        Dispatch tasks by number of resources
        Sort these tasks by length in Descending
        Map each task with the resource of equal Sequence
        End for
Step 3: For all resources DO
        Calc cost from equation (8)
        End for
```

5 Results

In order to evalute our algorithm, we built a program using **Matlab** because its easy to deal with matrices and in our algorithm all queues are matrices.Here, we present example that illustrate its work and we used a matrix as a performance measurment in order to evaluate the performance of the algorithm and to compare it with some of the traditional cloud scheduling algorithms

5.1 Performance Measurment

Depending on what scheduling performance is desired in the cloud, there exist different performance metrics for evaluating different scheduling algorithms. Here, the results are evaluated on the basis of **Makespan** performance measurement.

- **Makespan**: it is the time difference between the start and finish of the sequence of tasks. It can be calculated using the equation
Makespan = max (*Tcomp*$_i$); where ***Tcomp***$_i$ is the completion time of task (i).

5.2 Example

The aim of this example is to illustrate the basic functionality of the proposed algorithm. In this example, it is assumed that there is a cloud environment with two resources **R1**, **R2.** The processing speed of these resources and the cost of rent are shown in Table 1.

Resource	Processing speed (MIPS)	Cost
R1	1000	0.02
R2	2000	0.03

Table 1. Specification of the Resources

Also, assume we have a batch of ten tasks **T1**, **T2**..., **T10**, and the cloud manager is supposed to schedule all the tasks on the two available resources **R1** and **R2**. Table 2 represents the size details of both the instructions and data for all the tasks **T1** to **T10**.

Task ID	Task length in (MI)	Deadline
T1	2000	2
T2	3000	4
T3	1800	6
T4	1900	8
T5	3000	10
T6	500	12
T7	400	14
T8	5000	16
T9	1000	18
T10	4000	20

Table 2. Specification of the Tasks

After we applied our algorithm on these tasks we get result of first stage of our scheduling algorithm as shown in table 3.

Task sorted by priority	Task length in (MI)
T1	2000
T2	3000
T3	1800
T5	3000
T8	5000
T4	1900
T10	4000
T9	1000
T6	500
T7	400

Table 3. Stage one results

And for stage two result table 4 shows the mapping of tasks to the resources

Resource R1	Resource R2
T1	T2
T3	T5
T4	T8
T9	T10
T7	T6

Table 4. Stage two tasks mapping with resources

And for more detail we used Gantt chart to show the execution of tasks in the mapped resource as in figure 6.

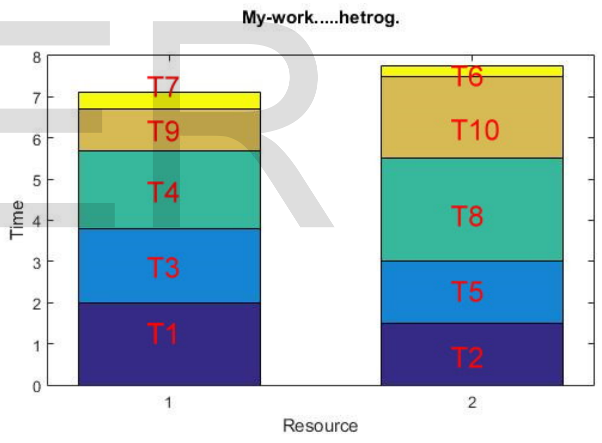

Fig 6. Execution of tasks

We make comparison with traditional algorithm like ***Earlist Deadline First*** with *First Fit* (**EDF-FF**) And ***Earlist Deadline First*** with *Best Fit* (**EDF-BF**) and the gantt chart of results is shown in figure 7 and 8 in Hetrogeneous and Homogeneous respectivley.

In Homogeneous environment we use:

R1=R2=1000 MIPS

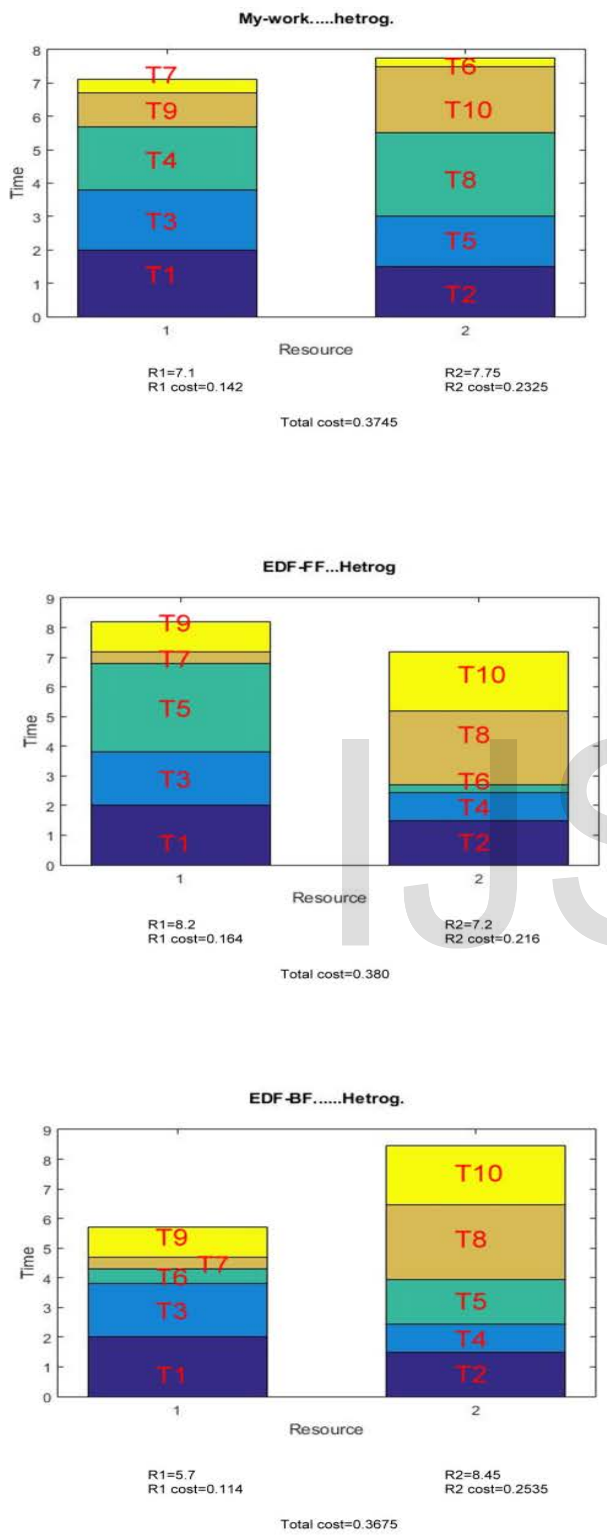

Fig 7. Gantt chart of our algorithm and EDF-FF and EDF-BF in heterogeneous environment

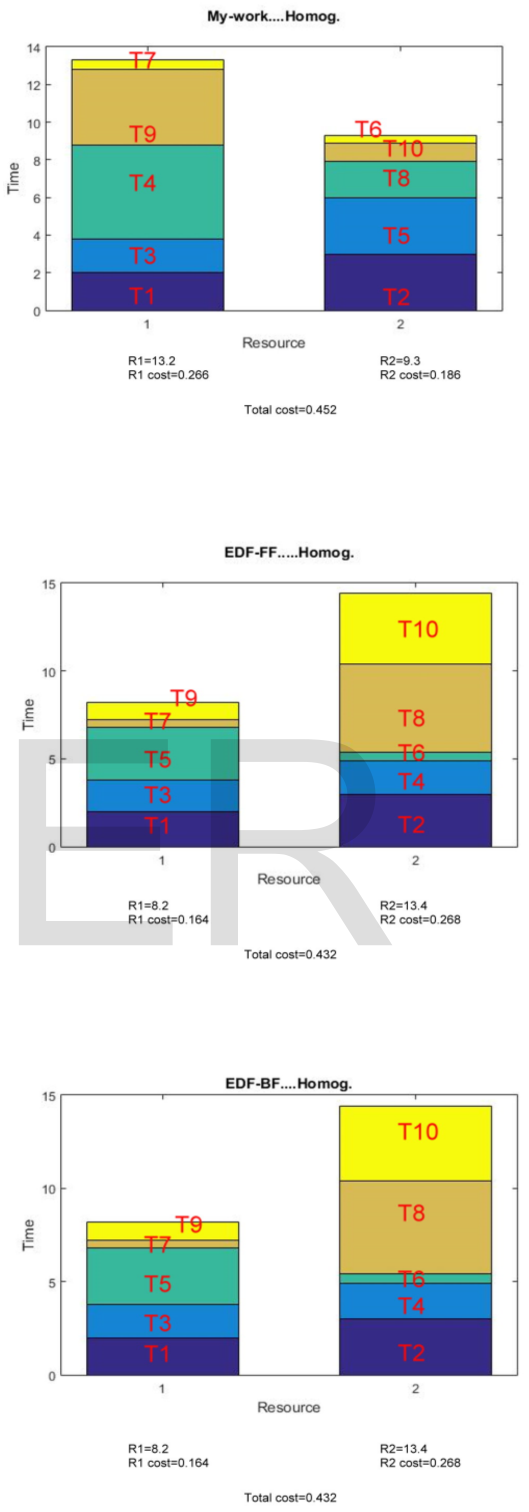

Fig 8. Gantt chart of our algorithm and EDF-FF and EDF-BF in homogeneous enviroment

And also we make comparison with **Max-Min** and **Min-Min** algorithms and the results are in figure 9 and 10 in Hetrogeneous and Homogeneous respectivley.

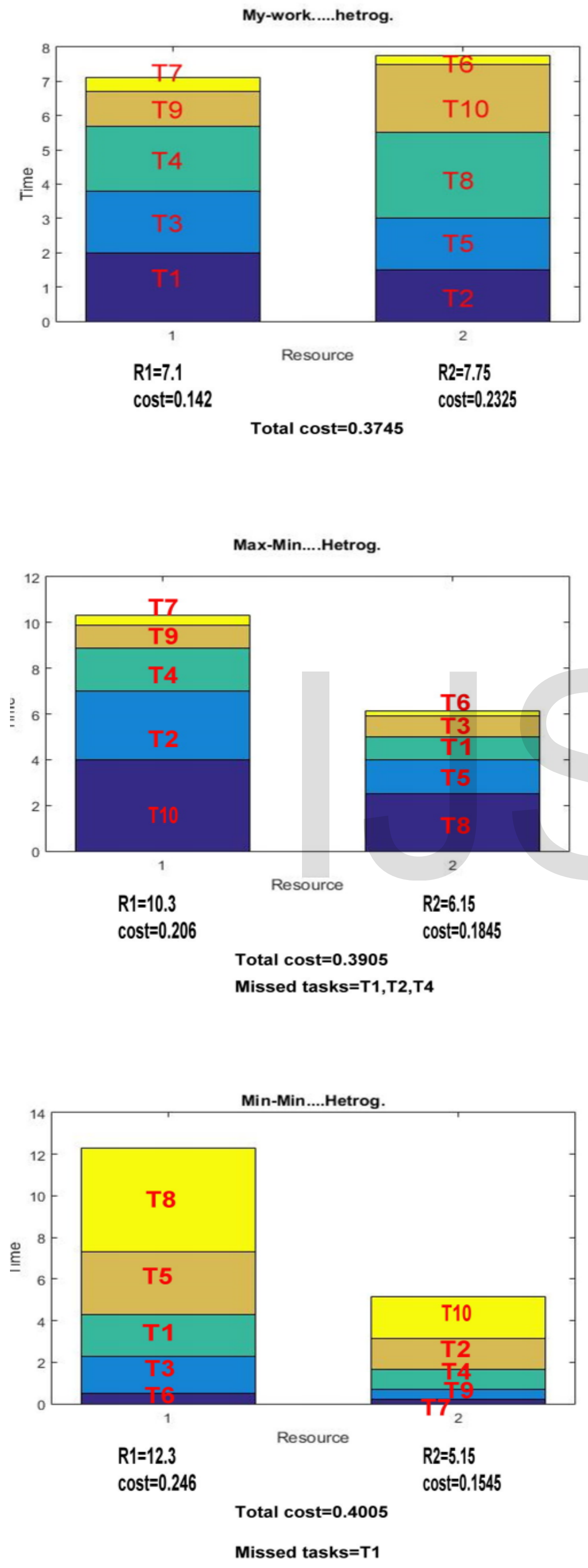

Fig 9. Gantt chart of our algorithm and Max-Min and Min-Min in heterogeneous enviroment

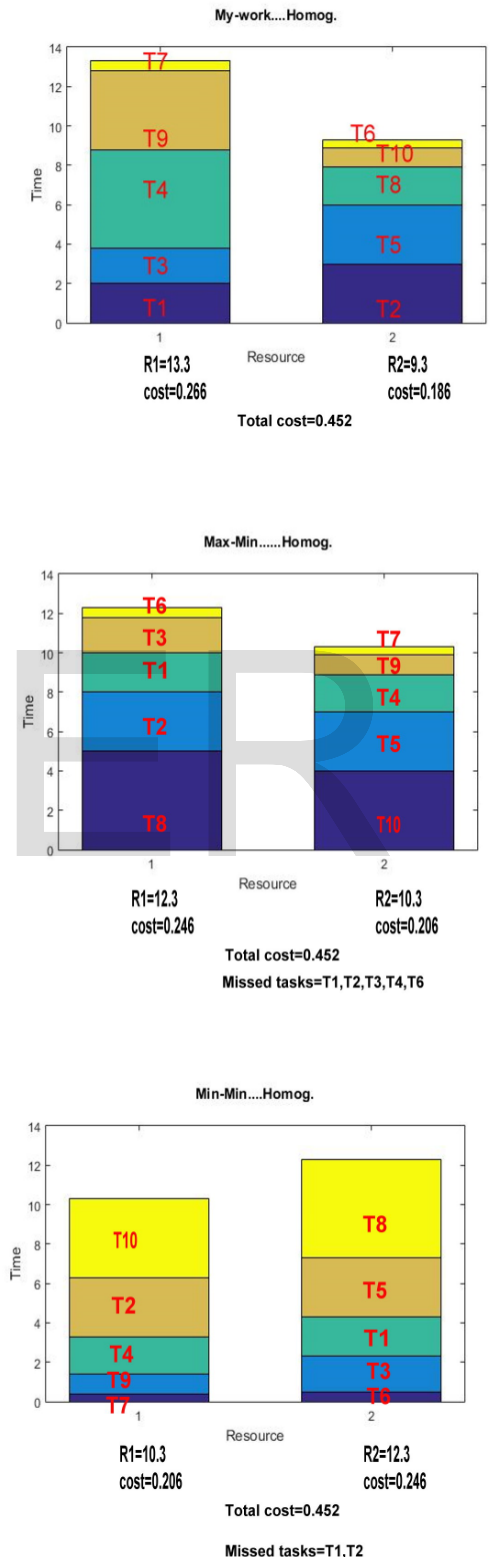

Fig 10. Gantt chart of our algorithm and Max-Min and Min-Min homogeneous enviroment

And the overall heterogeneous environment makespan results is shown in figure 11.

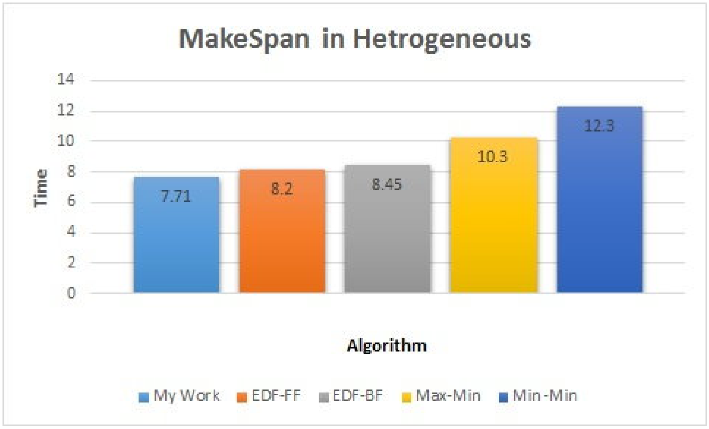

Fig 11. Makespan result in hetrogeneous environment

And for homogeneous environment makespan results is in figure 12

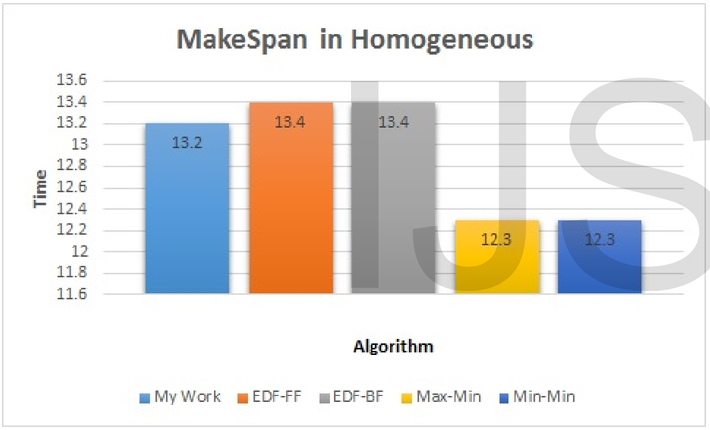

Fig 12. Makespan result in homogeneous environment

And for cost results in heterogeneous and homogeneous environment the result are shown in figure 13 and 14 respectively.

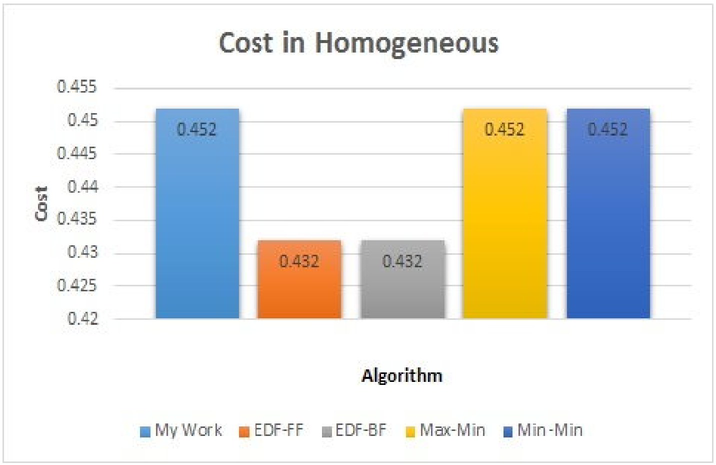

Fig 13. Cost results in homogeneous environment

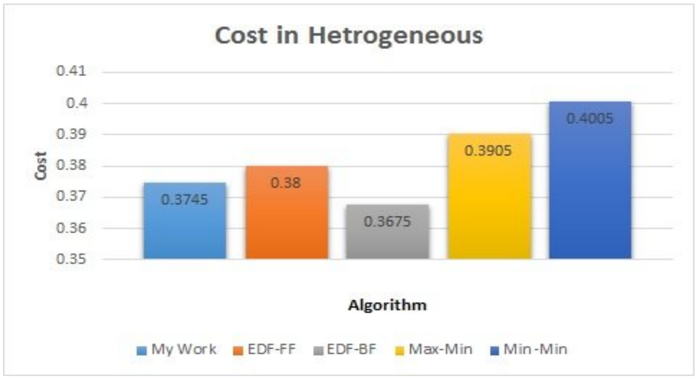

Fig14. Cost results in hetrogeneous environment

6. Conclusion and Future Work

The proposed research work examined the difficulties of batch task scheduling. The objectives of this work were to bolster the user satisfaction, to mitigate task violating its policy, to maximize resource utilization and consider the cost as important value. To achieve these objectives, our algorithm has been proposed for scheduling the batch tasks. The user satisfaction was achieved by neglecting the task that doesn't satisfy some condition. The number of tasks violating their deadline was reduced by filtering the tasks using multiple criteria. The priority was dynamically assigned to the accepted tasks in order to make good load balance. The prioritized tasks were efficiently mapped with VM either in homogeneous or in the heterogeneous environment and thereby efficiently balanced the load. The VM Scheduler has been deployed effectively scheduling the tasks. Our algorithm outperforms the existing scheduling algorithms by reducing the number of tasks violating their deadline that improves the user satisfaction. It also focused on load balancing that increases throughput and also resource utilization. In future, the work can be extended to develop an efficient cost and energy aware scheduler for processing both dependent and independent tasks.

Acknowledgment

I would like to thank Ministry of Science and Technology in Iraq for funding my M.Sc. study.

REFERENCES

[1] D. Bruneo, S. Distefano, F. Longo, and M. Scarpa. Stochastic evaluation of qos in service-based systems. *Parallel and Distributed Systems, IEEE Transactions on*, 24(10):2090–2099, Oct 2013.
[2] A. Verma, L. Cherkasova, and R.H. Campbell. Orchestrating an ensemble of mapreduce tasks for minimizing their makespan. *Dependable and Secure Computing, IEEE Transactions on*, 10(5):314–327, Sept 2013.
[3] Gunho Lee, Resource Allocation and Scheduling in Heterogeneous Cloud Environments, Spring 2012
[4] X. Tang, K. Li, Z. Zeng, and B. Veeravalli. A novel security-driven scheduling algorithm for precedence-constrained tasks in heterogeneous distributed systems. *Computers, IEEE Transactions on*, 60(7):1017–1029, July 2011.
[5] P. Choudhury, P.P. Chakrabarti, and R. Kumar. Online scheduling of dynamic task graphs with communication and contention for multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, 23(1):126–133, Jan 2012.
[6] P. Cichowski and J. Keller. Efficient and fault-tolerant static scheduling for grids. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 1439–1448, May 2013.
[7] T. Xie, X. Qin, and A. Sung. Sarec: a security-aware scheduling strategy for real-time applications on clusters. In *Parallel Processing, 2005. ICPP 2005. International Conference on*, pages 5–12, June 2005.
[8] K. M. Sim, “Agent-Based Cloud Computing,” *IEEE Transactions on Services Computing*,vol.5,no.4,pp.564-577,2012.
[9] Li. Chunxiao, A. Raghunathan and Niraj K. Jha, “A Trusted Virtual Machine in an Untrusted Management Environment”, *IEEE Transactions on Services Computing*, vol. 5, no.4,pp.472-483,2012
[10] Shelby Hyatt Funk, EDF Scheduling on Heterogeneous Multiprocessors, Chapel Hill 2004
[11] Bjorn B. Brandenburg, SCHEDULING AND LOCKING IN MULTIPROCESSOR REAL-TIME OPERATING SYSTEMS, Chapel Hill 2011
[13] Y. Zhang, H. Franke, et al., “An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration,” IEEE Transactions on Parallel and Distributed Systems, vol. 14, no. 3, pp. 236-247, 2003
[14] D. Carrera, M. Steinder, et al., “Autonomic Placement of Mixed Batch and Transactional Workloads,” IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 2, pp. 219-231, 2012.
[15] V. Gamini Abhaya, Z. Tari, et al., “Performance Analysis of EDF Scheduling in a Multi-Priority Preemptive M/G/1 Queue,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2149-2158, 2014.
[16] M. Stillwell, F. Vivien and H. casanova, “Dynamic Fractional Resource Scheduling versus Batch Scheduling,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 3, pp. 521-529, 2012.
[17] Etminani, K., Naghibzadeh, M.: A Weighted Mean Time Min-Min Max-Min Selective Scheduling Strategy for Independent Tasks on Grid. In Proceedings of the 2nd International Advance Computing Conference (IACC), IEEE Computer Society, Patiala, India, 4-9. (2010)
[18] Cao, Q., Wei. Z., Gong, W. M.: An Optimized Algorithm for Task Scheduling Based On Activity Based Costing in Cloud Computing. In Proceedings of the 3rd International Conference Bioinformatics and Biomedical Engineering (ICBBE). IEEE Computer Society, Beijing, China, 1-3. (2009).
[19] Sarada, N. S.: Enhanced Ant Colony System Based On RASA Algorithm In Grid Scheduling.International Journal of Computer Science and Information Technologies, Vol. 2. No.4, 1659-1674, 2011.
[20] Naseem, Mohammed: A Scheduling Algorithm to Enhance the Performance and the Cost of Cloud Services. Computer Engineering and Intelligent Systems, Vol.6, No.8, 2015
[21] C. Zhao, S. Zhang, et al., “Independent Tasks Scheduling Based on Genetic Algorithm in Cloud Computing,” *International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1-4, 2009.
[22] M. Kuanr, P. Mohanty, S. C. Moharana, “Grouping-Based Task Scheduling in Cloud computing using Ant Colony Framework,” *International Journal of Engineering Research and Applications*, 2013.