Statistical-Based Heuristic for Tasks Scheduling in Cloud Computing Environment

Ahmad Al-Qerem[1], Ala Hamarsheh[2]

[1]Department of CIS, Zarqa University,Zarqa, Jordan
[2]Faculty of Engineering and Information Technology, Arab American University, Jenin, Palestine

***Abstract*:** Cloud computing is an emerging and innovative technology that has taken the business information systems to wider extent with the fast sharing of vast web resources over the internet. It considers as an extension to distributed and parallel computing. Additionally, it enables sharing, organizing and aggregation of computational machines to satisfy the user demands. Utilizing the resources efficiently is the main challenge of cloud service provider. Task scheduling in cloud computing plays the main role in decreasing the execution time and cost and hence, increasing the profit. This paper addresses the problem independent tasks scheduling over different virtual machines in computational cloud environment. It introduces two batch mode heuristics algorithms for scheduling independent task: high mean absolute deviation first heuristic and QoS Guided Sufferage-HMADF heuristic. Besides, the paper presented other existing batch mode heuristics such as, Min-Min, Max-Min and Sufferage. The four heuristic modes are simulated and the experimental results are discussed using two performance measures, makespan and machine resource utilization.

***Keywords*:** Cloud Computing, Batch mode Heuristic, Scheduling Tasks, Makespan, Resource Utilization.

1. Introduction

Cloud computing has grown rapidly and received enormous interest since it offers flexibility and scalability to the users and organizations. Cloud computing is a huge distributed system which offers a pool of computing resources to cloud users using the internet. There are many organizations that provide cloud services and run on cloud computing environment such as IBM, Amazon, Google Engine, etc. These organizations services and resources to customers on the basis of pay per use at anytime from anywhere [1]. There are three main delivery models that offered by cloud computing. These models are Cloud computing offers three main delivery models which are Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). In Software as a Service (SaaS), Applications and access management tools are provided to users. Platform as a Service (PaaS) provides tools such as operating systems, databases, and network so consumers can install and develop their own software and applications. Infrastructure as a Service (IaaS) provides access to physical devices such as hardware and network so consumers can install and develop their own operating systems and applications [2]. Clouds in cloud computing is of several types based on the scalability and pooling up of the resources. Types are public, private, community, and hybrid clouds. Public clouds are available to the general public in a pay-as-yougo manner and they are owned by the cloud provider. Private clouds are operated only for a business or an organization and they are controlled by that organization or a third party. In community clouds, several organizations share the infrastructure of the cloud to support certain community that has common concerns. Hybrid clouds are combination of public, private, or community clouds [3]. Scheduling and allocation of resources and tasks are critical problems in cloud computing which many researches where carried out on it. Cloud providers must serve many consumers in cloud computing system. Therefore, scheduling is the major issue in establishing cloud computing system in order to reduce the execution time and the cost, thus maximizing resource utilization.

There are different types of clouds in which they are classified in terms of functionality. For example, computational cloud, data cloud, collaborative cloud and network cloud [4][5][6][7]. This paper addresses a computational cloud. The rest of the paper is organized as follows. Section two presents the related work. Section three proposes two batch mode heuristic: High Mean absolute Deviation First (HMADF) and QoS Guided Sufferage-HMADF. Section four discusses the simulation results. The conclusions and future work are presented in Section five.

2. Related Work

Cloud architecture explains the cloud construction. Besides, it describes the components of the cloud and how these components are interacted with each others. The architecture consists of upper layers and lower layers [11][12][13]. The former is a user-centric, while the later is a hardware-centric. Upper and lower layers consist of four sublayers. These sublayers are described as follows:

- Cloud Fabric Layer: it consists of several distributed resources. For example, PCs, networks, storage device, data sources and scientific instruments. The resources are represented in the form of clusters of PCs or racks of PCs, supercomputers, servers or workstations and regular PCs which run on different platforms. Scientific instruments such as seismographs (for recording earthquake), seismometer (for determining earthquake intensity), seismscope (for discovering earthquake), telescope and sensor networks offer real- time data that can be stored in a storage device or transmitted directly to computational sites.
- Core Middleware Layer: it provides several distributed services. For example, remote process management, information registration, security and QoS. Furthermore, it hides the complexity and heterogeneity in the fabric level.
- User-level Cloud Middleware Layer: it consists of programming tools such as libraries, compilers, and application development tools. Additionally, it utilizes the interfaces provided by lower level middleware (i.e. core

middleware) to provide higher levels of abstractions. Resource broker is responsible for managing, picking out and scheduling the tasks on machines.

- Applications and Portals Layer: it basically consists of engineering and scientific applications. For example, a grand-challenge problem such as LCH@home is required to allow the remotely access the data and computational power which they are needed to interact with scientific instruments and Cloud portals. Cloud portals provide web-enabled applications are used to users to submit tasks and then collect the results from particular machines.

Most of the task scheduling algorithms could effect on the users' tasks proficiency as well as in utilizing the resources efficiently particularly in IaaS cloud computing environment. Therefore, realizing the optimal distribution of users' tasks is still an unhandled issue for task scheduling in this environment, as shown in Fig 1. The task scheduling algorithms could be implemented in cloud computing environment as follows: Firstly, tasks and resources are mapped with respect to the current tasks and information of resources along with basic procedures. At that stage, tasks are mapped among the QoS conditions of cloud users and the resources are allocated to the application of the task to approve the competence of the task. The summary of the consequences is implemented by submitting the users' request [8][9][10][14].

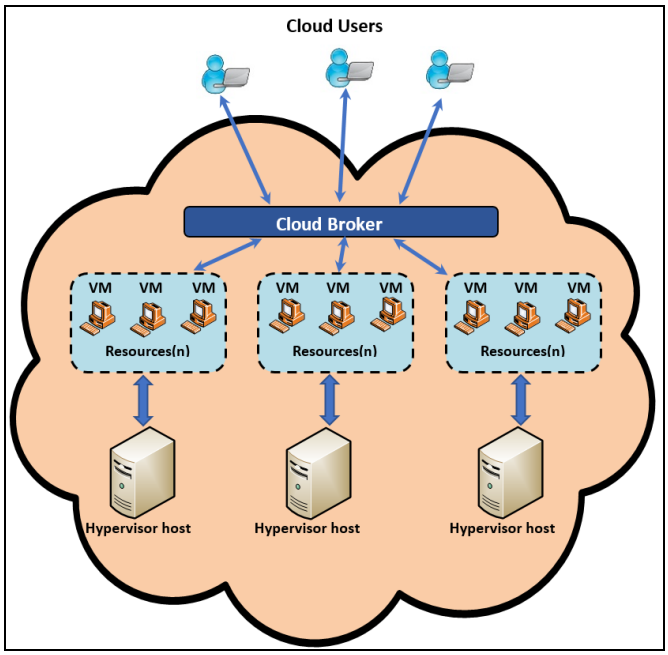

Figure 1. Task scheduling in cloud computing.

Cloud scheduling is a process of mapping Cloud jobs to resources (i.e. machines) [16]. A Cloud job can be divided into many small tasks. One of the main responsibilities of the scheduler is to select resources and scheduling jobs in such a way that the user and application requirements are accomplished, in terms of throughput and cost.

- Static versus Dynamic Scheduling: static scheduling refers to the fact that no new tasks, and tasks with high priorities, will be joined till all computations are ended. During computations, static scheduling requires that no job failures, and the resources have to be available all the time. In contrast, dynamic scheduling refers to that fact that new tasks cloud be joined during computations. Additionally, tasks high priorities cloud be processed within scheduling time. The schedule therefore changes over time.
- Immediate versus Batch Mode Scheduling: in immediate mode, tasks are scheduled one after another. Alternatively, this mode works as first come first served (i.e. tasks that arrived first will be computed first). Besides, if more than one task arrived simultaneously, the first one will be selected and not the best one. In contrast, the batch mode works as follows. Tasks are collected in batches and arrived at a time, one of the tasks is picked out from the batch. This mode does not consider the task arrival time and hence, a task that arrives first will have no priority. If the batch includes only one task, then the batch mode heuristic acts similar to immediate mode.
- Non-preemptive versus Preemptive Scheduling: in non-preemptive scheduling, when a task is assigned to a machine, it cannot be released before it completes the computation. A task that depends on the deadline has to wait until the computation is over even if it exceeds that deadline. In contrast, the preemptive scheduling works as follows. A task may be released before it completes the computation. In case of a high priority task has arrived, the current task checks the priority. If the current task priority is lower than the one of the arrived task, then the current task releases the machine. Otherwise, it continues until the computation is over.

This paper considers a static, batch and non-preemptive scheduling.

Current and previous research studies have primarily concentrated on heuristics based on different criteria. The previous work can be categorized into two types, immediate and batch mode heuristic. There are two types of heuristics in batch mode, QoS and non-QoS.

2.1 Non-QoS batch mode heuristic scheduling algorithms

In this section, four non-QoS batch mode heuristics will be explained.

- ***Min-Min***

This heuristic assigns highest priorities to tasks that have a minimum completion time (i.e. the task which can be completed sooner). Min-Min starts with a set of metatask in which it contains all unassigned tasks. After that, it computes the completion time of all tasks in metatask overall machines. Eq. 1 illustrates how the completion time is calculated. Min-Min has two phases. First, it finds the set of minimum expected completion time for each task in metatask (i.e. selects a suitable machine for each task that gives an earliest possible completion time). Second, a task with minimal expected completion time from MT will be selected and assigned to a corresponding machine. This heuristic requires O(m2n) time to assign the tasks to the machines [15][16].

Completion time = Execution time + Ready time (1)

Ready time of the machine It is also called machine availability, which is defined as the time required for a machine to complete all assigned tasks.

- ***Max-Min***

The Max-Min scheduling is very similar to Min-Min except in phase #2. The Max-Min scheduling assigns task with maximum expected completion time to a corresponding

machine. This heuristic requires O(m2n) time to assign the tasks to the machines [16][17].

- ***Sufferage***

This heuristic considers the sufferage value as a criteria that used to assign tasks to a machine. The Sufferage value is defined as the difference between the second earliest completion time and first earliest completion time. The basic idea behind this heuristic is that, a task should be assigned to a certain machine, and if it does not assign to that machine, the task will suffer [16].

- ***Segmented Min-Min***

This heuristic is considered as an improvement to Min-Min scheduling. Unlike Min-Min heuristic, this heuristic aims to perform large tasks before performing short ones which leads to improve the load balancing. For example, tasks are sorted into an ordered list by the average, minimum, or the maximum time that is expected for completion. Thereafter, the list is divided into segments with the equally sizes (i.e. the authors found through experiments that the optimum value of the number of segments is four). Then, apply Min-Min on each segment. The experiments show that this heuristic works better than Min-Min when length of tasks is dramatically different [21].

2.2 QoS batch mode heuristic scheduling algorithms

In this section, three QoS batch mode heuristics will be explained.

- ***QoS Guided Min-Min heuristic***

The traditional Min-Min heuristic does not consider QoS. In this context, QoS means bandwidth, speed, deadline, priority etc. [22]. In this heuristic, there are two levels of QoS: high QoS and low QoS. Tasks with a high QoS request is only assigned to high QoS machines. Alternatively, tasks with a low QoS request can be assigned to both low QoS and high QoS machines. The basic idea is that, heuristic maps the tasks that need high network bandwidth before the tasks that need low network bandwidth [22].

- ***QoS Sufferage Heuristic***

Similar to QoS guided min-min heuristic, QoS sufferage heuristic also divides the tasks into two groups, high QoS and low QoS. Then it maps both high QoS and Low QoS tasks by using sufferage heuristic [23].

- ***QoS Priority Grouping Heuristic***

In this heuristic, the tasks are distributed into two groups based on the number of available machines. Tasks that can be executed on all available machines are added to the low QoS group. Besides, tasks that cannot be executed on at least one machine are added to the high QoS group. Based on QoS level, it uses sufferage heuristic to assign the tasks to a machine [24]. Table 1 shows the significance of the QoS when some of QoS constrains are added to a particular heuristic.

Table 1. Summary of Heuristics with Their Objectives

Heuristics	Makespan	Resource Utilization	Load Balancing
MET	Yes	No	No
MCT	Yes	No	Yes
Min-Min	Yes	No	No
Max-Min	Yes	Yes	No
QoS Guided Min-Min heuristic	Yes	Yes	Yes
QoS priority grouping scheduling heuristic	Yes	Yes	Yes
Qos Suffrage heuristic	Yes	Yes	Yes

3. Proposed Heuristics

This paper presents two batch mode heuristics, High Mean Absolute Deviation First (HMADF) heuristic and QoS Guided Sufferage-HMADF heuristic.

3.1 Basic concepts and problem definition

The main items of the scheduling are tasks and machines. These items can be represented as a matrix in the form of two dimensional array. In this two-dimensional array, rows indicate tasks and columns indicate machines. Each entry in the matrix indicates the execution time of a task on a machine. A task is considered as a set of data or instructions. The data is measured in megabytes or in megabits, and the instructions are measured in million instruction unit (MI). Tasks can be one of two types, independent or dependent. The former means that tasks do not require to initiate connections with other tasks (i.e. no relationships between tasks). Thus, the task scheduling is done sequentially. It is with noting that tasks that don't have any dependency on each others are considered as Meta tasks. Hence, meta tasks can run in parallel [20]. On the other hand, the dependent tasks indicate that there is a dependency between tasks. For example, the output of one task can be used as input for other task(s).

Machines are considered as producers or service providers. Traditionally, these machines cloud be distributed geographically. Furthermore, they can be administered by different organizations or domains, and join or leave the cloud any time. Besides, the cloud have different characteristics and specifications. The scheduler assigns specific tasks these machines depending on the functional requirements that are provided by the end-user.

Several cloud scheduling heuristics are presented in recent years. The heuristics try to find a near optimal solution using matrices. There are three types of Matrices, consistent, inconsistent and semi-consistent. A matrix is considered to be consistent if the following condition is satisfied. If and only if a machine Mi takes a lower execution time to execute a task Ti than machine Mj, then the machine Mi always takes earliest execution time to execute any task Ti than machine Mj. In contrast, a matrix is considered to be inconsistent if the following condition is satisfied. If and only if a machine Mi takes a lower execution time to execute a task Ti than machine Mj, then the machine Mi may possibly take lower

execution time to execute any task Ti than machine Mj, or may not. A matrix is considered semi-consistent, if and only if a subset of an inconsistent matrix is consistent.

One of the cloud scheduling problems that this paper tried to handle is how to schedule m tasks on n machines with minimal makespan (i.e. the overall processing time). Additionally, the paper proposed solutions that are expected to improve the efficiency of the machines. The problem can be mathematically described as follows. Consider Ti (where i = 1,2,3,…….,m) as m independent tasks need to be scheduled, and Mj (where j=1,2,3,……n) as n are the available machines. Hence, m tasks and n machines are of m × n order. The Expected Execution Time (EET) for task Ti on machine Mj is Ei,j. The EET matrix is generated by the following methods that are described in [19]. The main goal is to find an efficient scheduling strategy SS that minimizes the overall processing time and maximizes machine utilization.

3.2 System Model

As shown in Fig. 2 the system model consists of four components: users, CMB, CRS and Machines. The system works as follows, it starts by allowing each user to submit a certain tasks. After that, the tasks are collected in a batch and sent directly to CMB. The responsibility of the CMB is to map tasks to available machines according to a certain scheduling method. This paper proposed a new scheduling method that used to map tasks to machines efficiently (see later). The last component in the system model is the CRS module, it provides information that are related to machines to CMB. Moreover, it is responsible for machine registration, machine directory management and status of the machine.

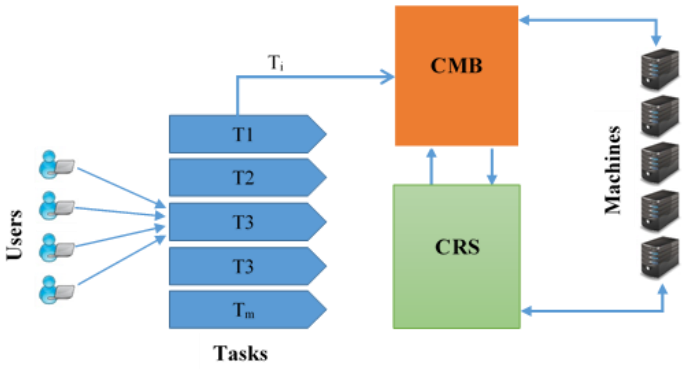

Figure 2. The system model

3.3 Notations

The following notations are used in the paper.

Notation	Definition
TQ	Task Queue
Ti	Task ID of task i
Mj	Machine ID of machine j
Ci,j	completion time of Ti on Mj
Rj	Ready time of machine j

3.4 High Mean Absolute Deviation First (HMADF) Heuristic

The current heuristics techniques use different decision factors in terms of the expected execution time (i.e. minimum, maximum, median or mean of the expected execution times). These factors don't consider the actual distribution of machines. Moreover, they don't take into account all values of the execution time for a certain task on all machines. As a result, the previous concerns can cause a wrong assignment of tasks to machines and hence, the completion time of these tasks will be effected negatively. Therefore, measuring the distribution and considering all values of data in EET matrix are needed. In order to alleviate this problem, the proposed technique allocates machines based on the highest median absolute deviation (MAD).

In this work, an improved task scheduling heuristic; high median absolute deviation first (HMADF) is proposed. It considers a new selection factor for task scheduling (i.e. the median absolute deviation of the expected execution time of tasks on all machines). The proposed technique uses MAD to calculate the average distance between each data value and the median. The basic idea behind this technique is that, tasks with greater median absolute deviation of completion time, will be assigned a higher priority than tasks with less median absolute deviation. Therefore, delaying the assignment of tasks with low MAD values will not affect the system performance (i.e. overall makespan). Alternatively, tasks with a greater median absolute deviation have more variations in their execution time when use different machines. Hence, if the assignment of these tasks is delayed, this will reduce the opportunity of mapping these tasks to faster machines. The simulation results show that the overall makespan will be improved when using this scenario.

3.4.1 Pseudo code: HMADF-heuristic

```
1. while TQ !=Null
2. for all meta-tasks Ti in TQ
3.       for all machines Mj
4.       compute  C i,j = Ei,j + Rj
5.       end for
6. end for
7. Compute the mean absolute deviation
   of each task
8. Find a task ti having the highest
   mean absolute deviation
9. Assign task ti to machine mj which
   gives the earliest completion time
10. Delete task ti from TQ
11. Update machine mj availability time
    (Rj)
12. Update completion time of all
    unmapped task(s)
13. end while
14. Compute "Makespan" and other
    performance measures
```

3.4.2 Description

The previous pseudo code first finds the absolute deviation of each task. Then, a task with the highest mean absolute deviation will be selected. Thereafter, a task with the highest mean absolute deviation will be assigned to a machine that gives the earliest completion time. This scenario will be repeated until no meta-tasks are present in TQ.

3.5 QoS Guided Sufferage-HMADF heuristic

The main goal of the currently scheduling algorithms, such as Min-Min, Max-Min and sufferage is to reduce the makespan. Such scheduling algorithms did not take into account the need for QoS, and hence, the overall scheduling performance

will be negatively affected [12]. The authors believe that the addition of QoS constraints to the current scheduling algorithms will improve their scheduling performance. There are some of QoS guided algorithms have considered only one-dimension of QoS parameters. For example, QoS guided min-min and QoS guided Sufferage. These algorithms consider the bandwidth as QoS parameter. However, they did not take into account other important parameters, such as, the availability of application tasks requirements. This can be calculated as the ratio of the total time that the computing resource is functioning during a given interval. Since the communication between machines in the cloud are unreliable and changed dynamically, add the availability constraint to QoS will be needed in cloud scheduling. This proposed heuristic implements both the bandwidth and the availability as QoS constraints in the scheduling algorithm to achieve a better performance.

The proposed heuristic considers binding between tasks with high QoS request and machines based on the traditional sufferage heuristic as this heuristic shows a better makespan among other batch mode heuristics [12]. Basically, not all machines can have the ability to run tasks with high QoS. Thus, neglecting this consideration when binding between high QoS tasks and machines decreases the number of machines that provide high QoS as these machines might be assigned to low QoS tasks. As a result, the overall makespan will be increased if high QoS tasks are assigned to low QoS machines. The sufferage heuristic overcomes this problem because it binds each high QoS task with high QoS machine.

For tasks with low QoS request, we considered the matching of these tasks and machine based on the proposed heuristic HMADF.

3.5.1 Pseudo code:QoS Guided Sufferage HMADF heuristic

```
1. while TQ !=Null
2. for all meta-tasks Ti in TQ
3.    for all machines Mj
4.      compute  C i,j = Ei,j + Rj
5.    end for
6. end for
7. do until all tasks with high QoS
   request in TQ are mapped
8. // Applying Sufferage
9. for each tasks with high QoS in TQ ,
   find a machine mj in the QoS
   qualified machine set that provides
   the earliest completion time
10. find the task Ti with the maximum
   sufferage value
11. assign task Ti to the machine mj
   that gives earliest completion time
12. delete task Ti from TQ
13. update machine mj availability time
14. update completion time of all
   unmapped task(s)
15. end do
16. do until all tasks with low QoS
   request in TQ are mapped
17. // Applying HMADF
18. for each tasks with low QoS in TQ ,
   find a machine mj that provides the
   earliest completion time
19. find the task Ti with the high mean
   absolute deviation
20. assign task Ti to the machine mj
   that gives earliest completion time
21. delete task Ti from TQ
22. update machine mj availability time
23. update completion time of all
   unmapped task(s)
24. end do
25. end while
26. Compute "Makespan" and other
   performance measures
```

3.5.2 Description

The previous pseudo code handles the scenario of matching between high QoS tasks' requests and machines based on the traditional Sufferage heuristic approach. Furthermore, the pseudo code handles the scenario of matching between low or non-QoS tasks' request and machines based on traditional Min-Min heuristic. The code starts by calculating and finding out the completion time of all tasks in Meta-task (MT) on all machines (from step# 1 to step# 6). After that, it divides the tasks in MT into two categories: high QoS and low QoS. It applies the Sufferage heuristic to tasks that have high QoS (from step# 7 to step# 14), these steps are repeated untill the mapping of all the high QoS tasks is done. Alternatively, it applies the HMAD heuristic to the remains tasks that have low QoS (from step #15 to step# 22).

4. Simulation and Results

This section analyzes the performance metrics (i.e. makespan and machine utilization) of proposed heuristics and compare them with the existing heuristics (see later). The simulator considers the data sets (or instance) that is presented by Braun et al. [14]. The main parameters of the data set are in the following general form: t_mmnn, where t indicates the types of matrices: consistent, inconsistent and semi-consistent. We adopted consistent, inconsistent and semi-consistent in our experimental studies. "mm" indicates the task heterogeneity which is defined as the average variation along the columns. Similarly, "nn" indicates the machine heterogeneity which is defined as the average variation along the rows [14]. Each type of heterogeneity (task and machine) is divided into categories: high (hi) and low (lo) heterogeneity. Therefore, each type of matrix includes four test sets. For example, hihi, hilo, lohi and lolo. As a result, we have twelve data sets. These are: c_hihi, c_hilo, c_lohi, c_lolo i_hihi, i_hilo, i_lohi, i_lolo, s_hihi, s_hilo, s_lohi, s_lolo. The size of data set is 512×16 that was used to evaluate HMADF heuristic performance. In order evaluate QoS Sufferage-HMADF heuristic, we have used 1024×32 dataset. Where the first value indicates the number of tasks and the second value indicates the number of machines. We have simulated the proposed heuristics algorithms using MATLAB R2013a version 8.1.0.604. Table 2 shows the simulation parameters of the data set.

Table 2. Simulation Parameters

Parameters		Value
No. of Tasks		512,1024
No. of Resources		16,32
Task Heterogeneity	Upper Bound- Φ_b	3000
	Lower Bound-	100
Resource Heterogeneity	Upper Bound- Φ_r	1000
	Lower Bound- Φ_r	10

Two performance measures are used to evaluate the heuristics, the makespan and resource utilization.

4.1 Makespan

The Makespan is defined as the maximum completion time that is required to assign all tasks to machines. This metric is important during the scheduling phase as it should be minimal as much as possible in order to achieve a better performance.

The makespan is computed as follows:

makespan = max (CT(ti, mj)) , where $i=1,2,...,M$, $j=1,2,....,N$ (2)

Correspondingly, it can be computed as the following:

$makespan = max\ (RT_j), j=1,2,....,N$ (3)

Here, RTj is the maximum computed machine ready time (i.e. the time when machine mj complete execution of all the assigned tasks).

4.2 Resource Utilization

Resource utilization is defined as the time that the machine is busy during the scheduling time.

The resource utilization of the jth resource is computed as follows:

$$ru_j = \frac{RTj}{makespan}, for\ j= 1,2,\ldots\ldots, N \quad (4)$$

The average resource utilization is:

$$\text{Avg-RU}=\frac{\sum_{j=1}^{N} ru\ j}{N} \quad (5)$$

The average resource utilization (in percentage) is:

$$\%\text{RU}=\frac{\sum_{j=1}^{N} mu\ j}{N}\times 100 \quad (6)$$

Fig. 3 and Fig. 4 show the makespan and machine utilization for the following heuristics: min-min, max-min, sufferage and the proposed HMADF heuristic. Fig. 3 shows that the proposed HMADF heuristic scheduling algorithm produces the smallest makespan values in the twelve types of ETC matrices. Thus, the HMADF produces the best makespan values among the existing batch mode heuristic scheduling algorithms. Fig. 4 illustrates the comparisons between HMADF heuristic and the existing heuristic scheduling algorithms in terms of resource utilization. We notice that the max-min heuristic gives better resource utilization values among the other heuristic scheduling algorithms.

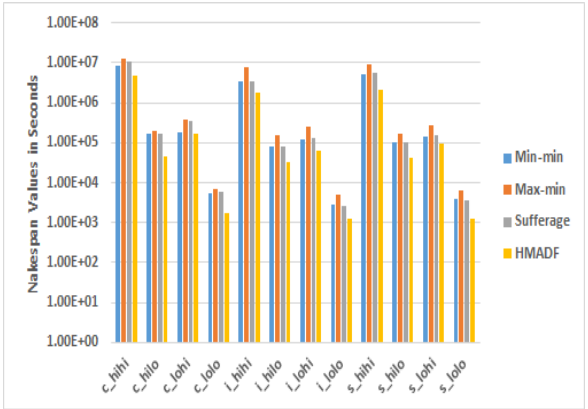

Figure 3. Comparison of makespan values obtained by Heuristics

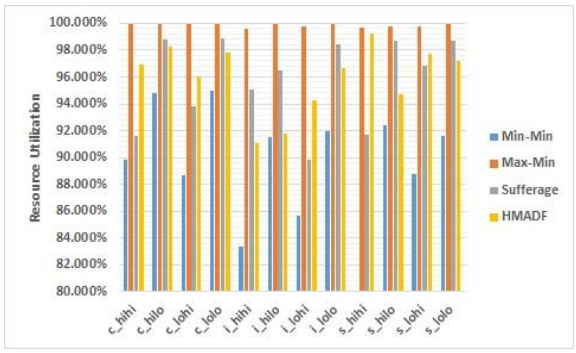

Figure 4. Comparison Results on Resource Utilization

The following scenario evaluates and compares the performance of both heuristics: sufferage-HMADF heuristic and guided min-min heuristic. All experiments are made using three distinct groups. These groups are classified based on the percentage of the number of high QoS tasks with respect to the number of low QoS machines. In each group, there are 1024 tasks and 32 machines. Group #1 consists of 30% high QoS tasks and need to be assigned to 70% low QoS machines. Group #2 consists of 50% high QoS tasks and need to be assigned to 50% low QoS machines. Group #3 consists of 70% high QoS tasks and need to be assigned to 30% low QoS machines.

Fig. 5, Fig. 6 and Fig. 7 illustrate the makespan comparison of the three groups using QoS guided Min-Min heuristic and the proposed heuristic QoS Guided Sufferage-HMADF. Additionally, Fig. 8, Fig. 9 and Fig. 10 illustrate the resource utilization comparison of the three groups using QoS guided Min-Min heuristic and the proposed heuristic QoS Guided Sufferage-HMADF.

Figure 5. Comparison of makespan values obtained by Heuristics over 1024x32 for Group1

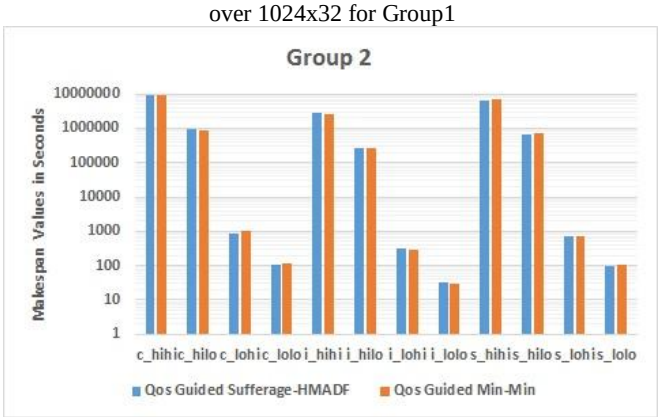

Figure 6. Comparison of makespan values obtained by Heuristics over 1024x32 for Group2

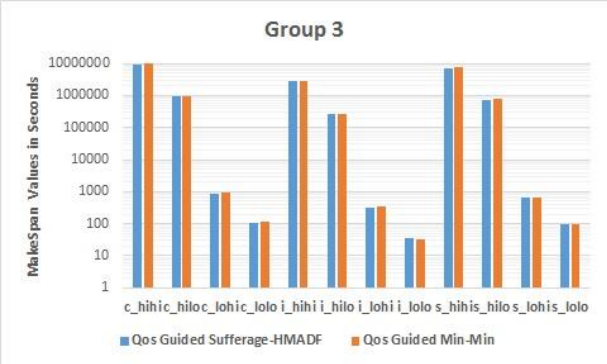

Figure 7. Comparison of makespan values obtained by Heuristics over 1024x32 for Group3

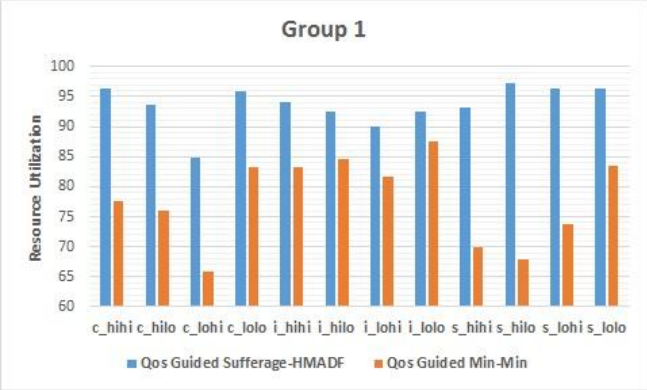

Figure 8. Comparison of Resource Utilization obtained by Heuristics over 1024x32 for Group1

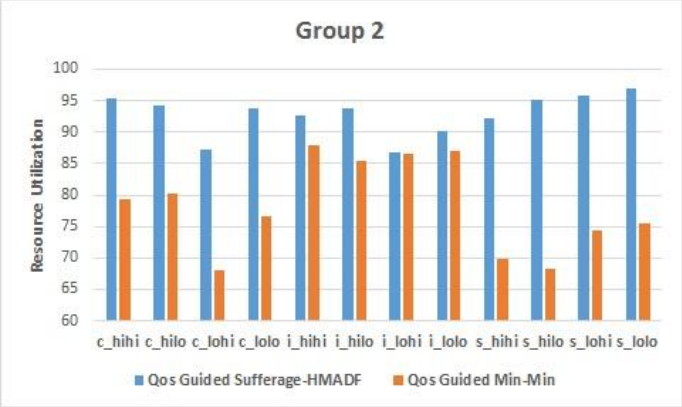

Figure 9. Comparison of Resource Utilization obtained by Heuristics over 1024x32 for Group2

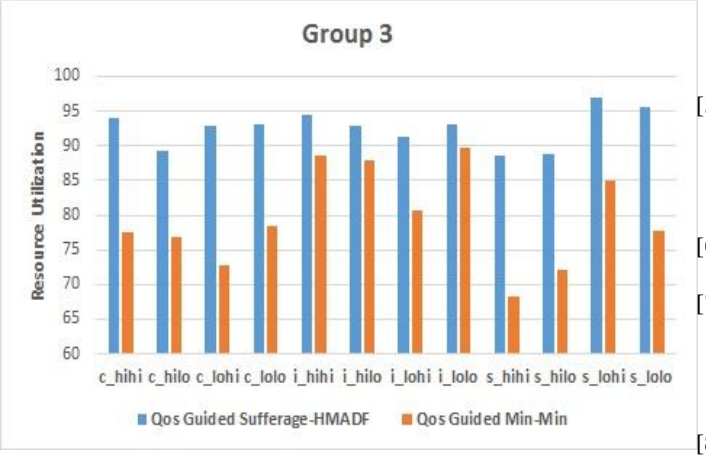

Figure 10. Comparison of Resource Utilization obtained by Heuristics over 1024x32 for Group3

In all experiments, the proposed QoS Guided Sufferage-HMADF heuristic achieves better results in makespan metric particularly in consistency and semi consistency instances for all groups. Moreover, the proposed heuristic achieves better results in the resource utilization metric in all groups over twelve instances. On the other hand, the QoS guided Min-Min heuristic shows better results particularly in inconsistency instances. The proposed heuristic gets the lowest averaged makespan values over twelve instances in each group.

5. Conclusion and Future work

This paper proposed two batch mode heuristics, HMADF heuristic and QoS Guided Sufferage-HMADF. The experimental results show that the HMADF heuristic achieves the best performance with respect to the makespan metric. Whereas, the Max-min heuristic performs well in the resource utilization metric. Alternately, the next proposed heuristic QoS Sufferage-HMADF exceeds the existing QoS guided Min-Min heuristic with respect to overall makespan and resource utilization metrics.

The future will go towards applying a deadline for task assignment. Furthermore, it will go towards improving the proposed scheduling approach by using a set of mechanisms such as, fault-tolerance, dynamic priority, predicative scheduling and security

Acknowledgment

This research is funded by the deanship of Research and Graduate Studies in Zarqa University /Jordan"

References

[1] V. Manglani, A. Jain, and V. Prasad, “Task Scheduling in Cloud Computing.,” Int. J. Adv. Res. Comput. Sci., vol. 8, no. 3, 2017.

[2] L. Liu and Z. Qiu, “A survey on virtual machine scheduling in cloud computing,” in 2016 2nd IEEE International Conference on Computer and Communications (ICCC), 2016, pp. 2717–2721.

[3] D. P. Chandrashekar, “Robust and fault-tolerant scheduling for scientific workflows in cloud computing environments.” University of Melbourne, Australia, 2015.

[4] Naga Raju Dasari, Saritha V, “Architecture for Fault Tolerance in Mobile Cloud Computing using Disease Resistance Approach”, International Journal of Communication Networks and Information Security (IJCNIS), Vol. 8, No. 2, 2016.

[5] Pradeep Singh Rawat, Anuj Kumar Yadav, Varun Barthwal, “Grid resource computing environment simulation using GridSim toolkit”, 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), 2015.

[6] I. Foster and C. Kesselman, “The Grid: Blueprint for a New Computing Infrastructure”, Morgan Kaufmann, 1999.

[7] Santhosh B; Manjaiah D H, “A hybrid AvgTask-Min and Max-Min algorithm for scheduling tasks in cloud computing”, 2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2015.

[8] M. Murshed, R. Buyya and David Abramson, “GridSim: A Toolkit for the Modeling and Simulation of Global Grids”, Technical Report, Monash University, Australia, 2001.

[9] Fatos Xhafa, Ajith Abraham, “Natures Heuristics for Scheduling Jobs on Computational Grids”, Future Generation

Computer Systems, Volume 26, Issue 4, April 2010, Pages 608–621.
[10] Marek R. Ogiela, Leonard Barolli, "New paradigms for information and services management in grid and pervasive computing", Future Generation Computer Systems, Volume 67, February 2017, Pages 227-229.
[11] Mieso K. Denko, Tao Sun, Isaac Woungang, "Trust management in ubiquitous computing: A Bayesian approach", Computer Communications, Volume 34, Issue 3, 15 March 2011, Pages 398-406.
[12] R. Buyya, High Performance Cluster Computing, Pearson Education, 2008.
[13] D. Zhu and J. Fan, An Introduction to Aggregation Grid, "Proceedings of the Second International Conference on Semantics", Knowledge, and Grid (SKG), IEEE, pp. 86 87, 2006.
[14] D. Zhu and J. Fan, "Aggregation Grid", Proceedings of the IEEE International Conference on Information Technology (ICIT), pp. 357 364, 2007.
[15] R. Buyya and S. Venugopal, "A Gentle Introduction to Grid Computing and Technologies", Computer Society of India Communications, Vol. 29, No. 1, pp. 9 19, 2005.
[16] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen and R. F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems", Journal of Parallel and Distributed Computing, Vol. 59, No. 2, pp. 107 131, 1999.
[17] Tracy D. Braun, Howard Jay Siegel, and Noah Beck, "Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-time Predictions, Stochastic-based robust dynamic resource allocation for independent tasks in a heterogeneous computing system", Journal of Parallel and Distributed Computing, Volume 97, Pages 96-111, 2016.
[18] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys and B. Yao, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", Journal of Parallel and Distributed Computing, Vol. 61, No. 6, pp. 810 - 837, 2001.
[19] F. Dong, J. Luo, L. Gao and L. Ge, "A Grid Task Scheduling Algorithm Based on QoS Priority Grouping", Proceedings of the Fifth International Conference on Grid and Cooperative Computing (GCC), IEEE, pp. 58 61, 2006.
[20] S. Abuelenin, "Trust Based Grid Batch Mode Scheduling Algorithms", The 8th International Conference on INFOrmatics and Systems (INFO), pp. 46 - 54, 2012.
[21] Qinma Kang, Hong He, "A novel discrete particle swarm optimization algorithm for meta-task assignment in heterogeneous computing systems", Microprocessors and Microsystems, Volume 35, Issue 1, Pages 10-17, February 2011.
[22] H. Xiaoshan, S. Xianhe and G. V. Laszewski, "QoS Guided Min-Min Heuristic for Grid Task Scheduling", Journal of Computer Science and Technology, Vol. 18, No. 4, pp. 442 - 451, 2003.
[23] E. U. Munir, J. Li and S. Shi, "QoS Sufferage Heuristic for Independent Task Scheduling in Grid", Information Technology Journal, Vol. 6, No. 8, pp. 1166 1170, 2007.
[24] F. Dong, J. Luo, L. Gao and L. Ge, "A Grid Task Scheduling Algorithm Based on QoS Priority Grouping", Proceedings of the Fifth International Conference on Grid and Cooperative Computing (GCC), IEEE, pp. 58 61, 2006.