© 2017 IJSRSET | Volume 3 | Issue 5 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Selective Algorithm for Task Scheduling in Cloud Computing

Zonayed Ahmed*[1], Tanveer Ahmed[2]
*[1,2]Department of Computer Science and Engineering, Stamford University Bangladesh, Dhaka, Bangladesh

ABSTRACT

The cloud infrastructure is an environment that provide dynamic services using very large scalable and virtualized resources. However, scheduling these services to the resources to ensure QoS (Quality of Serviec) is an NP-complete problem. To ensure maximum throughput, it is imperative for the task schedulers to implement an efficient algorithm. In this paper, a combination of two known algorithms max-min and min-min called selective algorithm is proposed. Selective algorithm focuses on using the advantages of both the algorithms. Experimental results show availability of load balancing in a heterogeneous cloud environment. Comparative analysis against other known scheduling algorithms (FCFS, SJF, Max-Min, Min-Min) also provides a shorter makespan on the cloud resources.

Keywords: Selective Algorithm, Min-Min, Max-Min, Meta Task Scheduling, Makespan, Cloud Computing

I. INTRODUCTION

Task scheduling is a mapping mechanism from user's tasks to the appropriate selection of resources and its execution. Compared with grid computing, cloud computing has many unique features including virtualization and flexibility. By using the technology of virtualization, all physical resources are virtualized and transparent for users. All users have their own virtual device, these devises do not interact with each other and they are created based on users' requirements. In addition, one or more virtual machines can run on a single host computer so that the utilization rate of resources has been effectively improved. The independence of users' application ensures the system's security of information and enhances the availability of service [1]. Supplying resources under the cloud computing environment is flexible, we increase or reduce the supplying of resources depends on users' demand. Because of these new features, grid computing, the original task scheduling mechanism, can't work effectively in cloud computing environments [2].

The task scheduling goals of Cloud computing is provide optimal tasks scheduling for users, and provide the entire cloud system throughput and QoS at the same time. Specific goals are load balance, quality of service (QoS), economic principle, optimal operation time and system throughput [3, 4].

Task scheduling algorithm is responsible for mapping jobs submitted to cloud environment onto available resources in such a way that the total response time, the makespan, is minimized [5]. Many task scheduling algorithms are applied by resources manager in distributed computing to optimally allocate resources to tasks [6]. While some of these algorithms try to minimize the total completion time. Where the minimization is not necessarily related to the execution time of each single task, but the aim is to minimize overall the completion time of all tasks [7].

II. RELATED WORKS

Many heuristics have been proposed to obtain semi-optimal match. Existing scheduling heuristics can be divided into two categories: **on-line mode** and **batch-mode**.

In the **on-line mode**, a task is mapped to a machine as soon as it arrives at the scheduler. Some heuristic instances of this category follow:

MET (Minimum Execution Time): MET assigns each task to the resource that performs it in the least amount of execution time, no matter whether this resource is available or not at that time [8].

OLB (Opportunistic Load Balancing): OLB assigns each task to the resource that becomes ready next, without considering the execution time of the task on that resource. When more than one resource becomes ready, one resource is arbitrarily chosen [7].

In the **batch-mode** heuristics, tasks are collected into a set called *meta-task* (MT). These sets are mapped at prescheduled times called mapping events. Some instances of this category are as follows:

Suffrage: Suffrage [7] is based on the idea that a task should be assigned to a certain resource and if it does not go to that resource, the most it will suffer.

Max-Min: Max-Min assigns task with maximum expected completion time to the corresponding resource [8].

The Max-Min algorithm is given below:

Step 1: For all submitted tasks in meta-task Ti
Step 2: For all resource Rj
Step 3: Compute $C_{ij} = E_{ij} + r_j$
Step 4: While meta-task is not empty
Step 5: Find the task T_m consumes maximum completion time.
Step 6: Assign task T_m to the resource R_j with minimum execution time.
Step 7: Remove the task T_m from meta-tasks set
Step 8: Update r_j for selected R_j
Step 9: Update C_{ij} for all T_i

Assume that we have m Resources R_j ($R_1, R_2, .., R_m$) and we process n tasks T_i($T_1, T_2, ..., T_n$) to be mapped on these resources. Also expected execution time E_{ij} of task T_i on resource R_j is defined as required time of resource R_j to finish task T_i provided that R_j has no load when assignment occurs.

On the other side, expected completion time C_{ij} of task T_i on resource R_j is defined as the overall time consumption till finishing any assigned task previously assigned. Assume r_j denote the beginning of execution task T_i. From previous mentions, it can be concluded that $C_{ij} = E_{ij} + r_j$.

The makespan of complete schedule is defined as Max (C_i) where C_i is the completion time for a task T_i [5].

As shown, task T_m with maximum expected completion time is chosen to be assigned for corresponding resource R_j that gives minimum execution time.

Makespan is defined as a measure of the throughput of the heterogeneous computing system; like the Cloud Computing environment [8], [10].
Min-Min: Min-Min assigns task with minimum expected completion time to the corresponding resource.
[8].

QoS Guided Min-Min: QoS Guided Min-Min shown in [9] adds a QoS constraint (QoS for a network by its bandwidth) to basic Min-Min heuristic. Its basic idea is that some tasks may require high network bandwidth, whereas others can be satisfied with low network bandwidth, so it assigns tasks with high QoS request first according to Min-Min heuristic.

QoS priority grouping scheduling: Similar to QoS guided Min-min, new algorithm called QoS priority grouping scheduling that is proposed by F. Dong et al [11]. QoS priority grouping scheduling algorithm considers deadline and acceptation rate of the tasks and makespan of the whole system as major factors for task scheduling. It achieves better acceptance rate and completion time for submitted tasks compared with Min-min and QoS guided Min-min.

Segmented Min-Min: In Segmented Min-Min heuristic described in [12] tasks are first ordered by their expected completion times. Then the ordered sequence is segmented and finally it applies Min-Min to these segments. This heuristic works better than Min-Min when length of tasks are dramatically different by giving a chance to longer tasks to be executed earlier than where the original Min-Min is adopted.

Improved Max-Min: In Improved Max-min algorithm largest job is selected and assigned to the resource which gives minimum completion time [13].

Enhanced Max-Min: Here, A task just greater than average execution time is selected and assigned to the resource which gives minimum completion time [14].

The organization of this paper is as follows. In Section 3 (**Selective Algorithm**), detailed explanation of any modifications of max-min will be provided. In Section 4

(**Implementation and Experiments**), we will present the implementation of our algorithm through CloudSim and analysis of our findings. Discussed in Section 4(**Conclusion**) a summary of our full work as well as concerns to address for the future.

III. Selective Algorithm

Max-min algorithm allocates task T_i on the resource R_j where large tasks have highest priority rather than smaller tasks. For example, if we have one long task, the Max-min could execute many short tasks concurrently while executing large one. The total makespan, in this case is determined by the execution of long task. But if meta-tasks contains tasks have relatively different completion time and execution time, the makespan is not determined by one of submitted tasks. It would be similar to the Min-min makespan. For these cases, original Max-min algorithm losses some of its major advantages in large scale distributed environment. We can't use the Max-min and wait submitted tasks to decide what would be the allocation map, makespan, load balance, etc. We try to minimize waiting time of short jobs through assigning large tasks to be executed by slower resources. On the other hand execute small tasks concurrently on fastest resource to finish large number of tasks during finalizing at least one large task on slower resource.

So, max-min algorithm works better than min-min when there are many short tasks and few long tasks.

Table 1 is an example where max-min outperforms min-min. Here makespan of max-min is 40 but makespan of min-min is 53.

TABLE I
MAX-MIN OUTPERFORMS MIN-MIN

	R_0	R_1
T_1	2	3
T_2	3	4.5
T_3	8	12
T_4	40	60

Here each value indicates a particular task T_i's expected time E_{ij} on Resource R_j.

Again, Min-min is perfect in executing shorter tasks and doesn't have much advantage in executing larger tasks.

Table 2 is an example where min-min outperforms max-min.

TABLE IIIII
MIN-MIN OUTPERFORMS MAX-MIN

	R_0	R_1
T_1	4	8
T_2	4	8
T_3	6	12
T_4	6	12

Here makespan of max-min is 28 while makespan of min-min is 24.

Now, the pseudo code of the selective algorithm is as follows:

(1) Sort tasks in meta-task *MT* ascending.
(2) **While** there are tasks in *MT*
(3) **for** all tasks t_i in *MT*
(4) **for** all machines m_j
(5) Calculate CT_{ij}
(6) **for** all tasks t_i in *MT*
(7) Find minimum CT_{ij} and resource m_j that finds it
(8) **If** there is more than one resource that obtains the resource
(9) Select resource that will have less CT_{ij} after the task
(10) Calculate standard deviation (*sd*).
(11) Find place *p* in *MT* where difference of two consecutive CT_{ij} is more than *sd*.
(12) **If** p is in the first half of the sorted MT or sd is less than average CT_{ij}
(13) **then** assign min-min for next task
(14) **else**
(15) choose max-min
(15) delete assigned task from MT
(16) **End While**

So, the cloudlets are sorted first. Then, we calculate execution time for every cloudlet. As this is batchwise execution, this is very easy. Then we take waiting time to consideration and calculate completion time. In the second for loop, similar to the first phase of Min-Min and Max-Min, it finds minimum expected completion time (such that the task t_i has earliest expected completion time on machine m_j) of each task in MT, and the resource that obtains it, lines 6 and 7 in the algorithm. If there is more than one resource that obtains this minimum, we choose the resource based on which resource gives less completion time after the task.

Then we choose between max-min and min-min algorithm. We calculate *sd* and see where the consecutive two place have lesser sd. Or, if such a case doesn't happen, to provide a threshold we can assign average completion time.

There can be two cases:
a. If *sd* is less than a certain threshold, it means the length of all tasks are in a small range, so we will select Min-Min to assign the next task.
b. Otherwise, we will select Max-Min to assign the next task.

After assignment of a task to a corresponding resource, this task will be deleted from MT and the process will be repeated until all tasks will be assigned.

The flow chart of the algorithm is given below:

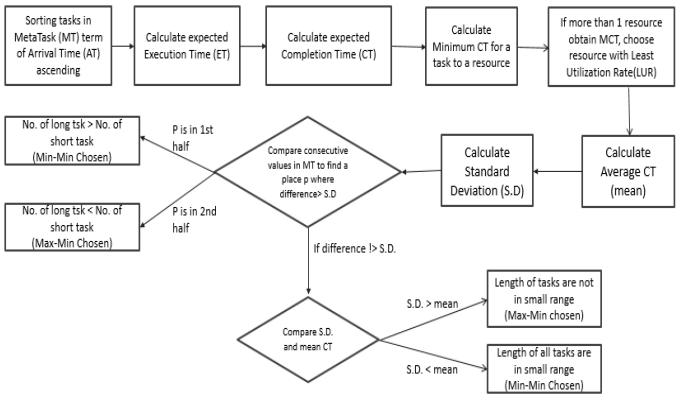

Figure 1: Flowchart of Selective Algorithm

IV. Implementation and Experiment

To implement the algorithm we take the following scenario:

We assume that minimum CT_{ij} for each task is found and S.D. is calculated too (i.e. lines 1 to 11 of the algorithm is executed). There is one short task and five long tasks:

Index	1	2	3	4	5	6
CT_{ij}	2	400	200	215	300	500

p = 1< s/2= 1<3

S = 6
S.D. = 158.35

Figure 2: Example of selective algorithm: where we choose min-min to select the next task

So, task lengths and completion time are shown in Figure 2. It can also been understood that both the parameters of the algorithm were satisfied. So, we choose min-min algorithm.

A reverse approach can also be seen in the following Figure 3:

Index	1	2	3	4	5	6
CT_{ij}	10	20	30	50	60	1000

S = 6
S.D. = 360.4

p = 5> s/2= 5>3

Figure 3: Example of selective algorithm: where we choose max-min to select the next task

In this case, there exists one long task and five short tasks. So, Max-Min outperforms Min-Min. As it can be seen, occurrence of the place of difference is in the second half, so Max-Min is selected to assign next task. Now, we have used JAVA language in Netbeans IDE and CloudSim [15] platform to create simulation environment.

Now, the resources and cloudlet specifications are given below:

TABLE IVVVI
PROBLEM SAMPLES RESOURCES SPECIFICATION

Problem Sample	Resource	MIPS	MBBS
P1	R1	50	100
	R2	100	5
P2	R1	150	300
	R2	300	15
P3	R1	300	300
	R2	30	15

TABLE VIIV
PROBLEM SAMPLES TASKS SPECIFICATION

Problem Sample	Resource	MI	MB
P1	T1	128	44
	T2	69	62
	T3	218	94
	T4	21	59
P2	T1	256	88
	T2	35	31
	T3	327	96
	T4	210	590
P3	T1	20	88
	T2	350	31
	T3	207	100
	T4	21	50

TABLE V
MAKESPAN OF DIFFERENT ALGORITHMS

Problem Sample	FCFS	SJF	Max-Min	Min-Min	Selective
P1	2.88	2.39	5.97	5.97	5.97
P2	5.63	5.47	4.7	3.22	3.22
P3	3.71	5.57	2.59	2.59	2.59

To compare the proposed algorithm with its two basic heuristics in time complexity measure, we computed the time complexity of Selective algorithm here.

In Line 1, an array of length *s* is being sorted. By using any sorting algorithm, it takes $O(s^2)$ time in its worst case.

In lines 3-5, two nested **for** loops takes *O(s.m)* time: internal **for** loop runs *m* times (number of machines) and external **for** loop runs *s* times (number of tasks).

Finding minimum CT_{ij} takes *O(m)* time (line 7). Finding the machine with minimum resource utilization rate in wits worst case, when all machines have same CT_{ij} for task t_i, takes O(m) time. This is done for all tasks (lines 7-9), so it takes *O(s.m)* time.

Computing standard deviation (line 10) consisted of calculating the average of *s* numbers: average of the array of CT_{ij} s and average of the array of $(CT_{ij})^2$ s both with *s* members. So it takes *O(s)* time.

Finding the place *p* in a list with *s* members (line 11) needs *O(s)* time. It is a sequential search.

Selection part of the new algorithm (lines 12-15) takes *O(1)* time, because the list is sorted and one should go to the start (for Min-Min) or the end (for Max-Min) of the list and no need to find minimum or maximum.

Deleting the assigned task, Line 16, takes *O(1)*, too, because the list is sorted and the task is deleted from the start or the end. Therefore, time complexity of lines 3-16 is the maximum of *O(sm), O(sm), O(s), O(s), O(1),* and *O(1),* that is *O(sm).*

This process, lines 3-16, is done for all tasks in MT; i.e. runs *s* times. Therefore, lines 2-17 takes $O(s^2m)$ time.

Consequently, time complexity of the Selective algorithm is:

$$\max(O(s^2),O(s^2m)) = O(s^2m)$$

Comparing it to Max-Min and Min-Min, the new heuristic does not impose any extra load and has the same time complexity as them.

V. CONCLUSION

Better throughput and load balancing has been achieved through Selective algorithm compared to different known batch mode scheduling algorithms. The algorithm covers all the advantages of both max-min and min-min while discarding the disadvantages. Here two heuristics: standard deviation and average completion time has been used for the algorithm. Other heuristics such as priority or ending time can be taken to make the algorithm more efficient.

VI.REFERENCES

[1] Zhexi, Y.A.N.G. and Huacheng, X.U.E. 2012. Informatization Expectation with Cloud Computing in China. *Indonesian Journal of Electrical Engineering and Computer Science*, *10*(4), pp.876-882.

[2] Liu, J., Luo, X.G., Li, B.N., Zhang, X.M. and Zhang, F., 2013. An intelligent job scheduling system for web service in cloud computing. *Indonesian Journal of Electrical Engineering and Computer Science, 11*(6), pp.2956-2961.

[3] You, X., Chang, G. and Deng, X., 2006. et. Grid Task Scheduling Algorithm Based on Merit Function. *Computer Science*, *33*(6).

[4] Yao, W., Li, B. and You, J., 2002. Genetic scheduling on minimal processing elements in the grid. *AI 2002: Advances in Artificial Intelligence*, pp.465-476.

[5] Parsa, S. and Entezari-Maleki, R., 2009. RASA: A new task scheduling algorithm in grid environment. *World Applied sciences journal*, *7*(Special issue of Computer & IT), pp.152-160.

[6] Chunlin, L. and Layuan, L., 2006. QoS based resource scheduling by computational economy in computational grid. *Information Processing Letters*, *98*(3), pp.119-126.

[7] Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D. and Freund, R.F., 1999. Dynamic mapping of a class of independent tasks onto heterogeneous

computing systems. *Journal of parallel and distributed computing*, *59*(2), pp.107-131.

[8] Freund, R.F., Gherrity, M., Ambrosius, S., Campbell, M., Halderman, M., Hensgen, D., Keith, E., Kidd, T., Kussow, M., Lima, J.D. and Mirabile, F., 1998, March. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In *Heterogeneous Computing Workshop, 1998.(HCW 98) Proceedings. 1998 Seventh* (pp. 184-199). IEEE.

[9] He, X., Sun, X. and Von Laszewski, G., 2003. QoS guided min-min heuristic for grid task scheduling. *Journal of Computer Science and Technology*, *18*(4), pp.442-451.

[10] Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D. and Freund, R.F., 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, *61*(6), pp.810-837.

[11] Dong, F., Luo, J., Gao, L. and Ge, L., 2006, October. A grid task scheduling algorithm based on QoS priority grouping. In *Grid and Cooperative Computing, 2006. GCC 2006. Fifth International Conference* (pp. 58-61). IEEE.

[12] Wu, M.Y., Shu, W. and Zhang, H., 2000. Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th* (pp. 375-385). IEEE.

[13] Elzeki, O.M., Reshad, M.Z. and Elsoud, M.A., 2012. Improved max-min algorithm in cloud computing. *International Journal of Computer Applications*, *50*(12).

[14] Bhoi, U. and Ramanuj, P.N., 2013. Enhanced max-min task scheduling algorithm in cloud computing. *International Journal of Application or Innovation in Engineering and Management (IJAIEM)*, *2*(4), pp.259-264.

[15] Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A. and Buyya, R., 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, *41*(1), pp.23-50.