Journal of Theoretical and Applied Information Technology
15th May 2021. Vol.99. No 9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

QPSJF: A NOVEL CLOUD COMPUTING TASK SCHEDULING POLICY BASED ON COMBINATION OF SCHEDULING ELEMENTS

[1]AMEERA JARADAT, [2]AHMAD AL-OMARI

[1, 2] Computer Science Department, Faculty of Information Technology & Computer Sciences, Yarmouk University, Irbid-Jordan

E-mail: [1]ameera@yu.edu.jo, [2]ahmad.alomari56@yahoo.com

ABSTRACT

Cloud computing is considered one of the rapidly emerging computing environments. It gives the user the ability to choose among the many computing and storage services. Service providers provide the services to users. Users prefer to select an appropriate datacenter for their requests to satisfy their requirements. The need for efficient task scheduling in a cloud computing environment to improve cloud performance motivated researchers to investigate existing task scheduling algorithms and/or to improve existing ones and to develop new ones. This article proposes a task scheduling approach named Queue Priority Shortest Job First scheduling (QPSJF), which is effective in optimizing execution time, waiting time, and response time. The proposed QPSJF algorithm distributes the cloudlets over three designated queues according to their length and priority. The efficiency of the proposed algorithm is supported through simulation and comparative analysis.

Keywords: *cloud computing; Task scheduling Algorithm; Virtual Machine; CloudSim; Shortest Job First; Priority Queue.*

1.0 INTRODUCTION

Cloud computing is considered as one of the most important and essential ingredients of the present and future computing environment that produced a pronounced effect in the information technology sector. It refers to an integrated environment that provides the users with several on-demand system resources including storage, computing resources, and computing power [1].

The cloud environment focuses on illustrating the availability of data centers (DCs) to provide cloud computing services to fulfill the computational and storage requirements of customers. With the new advancements in cloud computing, data access and resource usage from the customer side becomes easier and cheaper because of its zero-cost infrastructure, and all of these needs will be found in one place reachable by all users of the cloud.

Currently, there are five major types of cloud computing environments based on the services provided and users' scope according to [2] and [10] as follows:

a. Private clouds (also called internal clouds): This type of cloud is designed to be used by one organization and it is built from its organization infrastructure.

b. Public clouds: Public clouds are used by more than one organization and are publicly available over the internet such as Amazon AWS, which is considered one of the largest public clouds available. This type of cloud provides services to customers by cloud service providers.

c. Hybrid clouds: This type of cloud is a combination of private clouds and public clouds. In this type, an organization uses public clouds for insensitive information and private clouds for sensitive ones.

d. Virtual private clouds (VPCs): It is an on-demand pool of computing resources designed to avoid the limitations of private cloud needs. In VPCs, a private cloud is built upon a public cloud by using the infrastructure service from the cloud to formulate a private cloud with a customized architectural design. A good example of VPC in the Amazon VPC launched in 2009.

e. Community clouds: In this type of cloud, a computing environment is shared among several organizations or a group of people with

common needs and requirements. Examples of community clouds include banks, a group of companies working on the same platform, Google Apps for Government, and Microsoft Government Community Cloud.

The cloud-computing environment is generally integrated with several network architectures such as peer-to-peer architecture, client-server architecture, grid computing architecture, and utility computing. The peer-to-peer architecture enables two distributed devices to communicate with each other as peers and each peer may act as a client or server at the same time. The client-server architecture can be defined as an architecture that consists of two major parts, the clients and the server. A client sends its requests to a server asking for a specific service. On the other hand, the server responds with the requested service. Grid computing is a kind of parallel computing that allows a cluster of computers to work together to solve a large number of tasks. Utility computing is concerned with providing customers with packages of services to satisfy their computational needs.

Cloud services are provided in a pay-per-use manner and are available in three main models namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). In the IaaS model, the users are provided with online-virtualized services that have high capabilities in Application Programming Interfaces (API). These services are publicly available over the internet for direct use such as servers, load balancers, and virtual machines. The second model (i.e., PaaS) provides users with a full-suited framework to design and customize their software and applications. It is a cost-effective model, easy to use, and it makes the development operation as fast as possible for satisfying the user's needs. The third model, (i.e., SaaS) is a distributed model that allows customers to use applications and services hosted by a third party.

In addition to the above-mentioned three models, there are three more models namely: The mobile backend as a service (MBaaS), the server-less computing model, and the function as a service (FaaS) model. The first (i.e., the MBaaS) computing model offers cloud storage and high-level APIs for web and mobile applications developers to minimize the programming efforts. The second (i.e., the server-less) computing model is a cloud code execution model that offers an environment to execute codes, as the codes cannot be executed without a server, and manage virtual machines. The third model (FaaS) is a service remote procedure call hosted on a server to allow the developers to develop a responded events function.

Cloud computing has come to facilitate many computing processes, by offering many tools and features [4]. The cloud computing environment offers many features and tools to simplify and enhance the task secluding process. Cloudsim toolkit is a very powerful framework provided by cloud computing. It can be used to simulate the behavior of the task scheduling process by providing programmers with an integrated environment that allows researchers to develop and create their scheduling algorithms or use existing ones [5]. CloudSim can be integrated with Java programming language as a full package to be used for several tasks such as the task scheduling process. CloudSim allows researchers to create tasks that are called cloudlets, virtual machines (VMs) to process these cloudlets and create datacentres (DCs) to manipulate and run the task scheduling algorithms [6].

Task scheduling is the process of assigning computing recourses, mainly processor, to the various processes. Task scheduling is a main issue in cloud computing since it handles the allocation of cloud rescores over variety of cloud users. Task scheduling plays an important role in ensuring the quality of service in the cloud-computing environment. Quality of service takes account of parameters like execution time, waiting time, makespan, in addition to some other parameters. It is important to select the task scheduling algorithm that enhances the cloud computing performance [24].

The task scheduling problem is considered as one of the most important problems that affect the performance of a given computing environment and may limit its behavior [10]. There exist many task scheduling algorithms. Most of these algorithms focus on minimizing four main factors that affect the performance of the task scheduling operation. These factors are the makespan, the execution time, the waiting time, and the energy consummation [3]. A task scheduling algorithm can be defined as an algorithm that intends to schedule tasks in an efficient way that minimizes one or more of these factors.

The classical scheduling algorithms like SJF and priority have some problems. In priority scheduling low priority processes may never execute if higher priority processes keep arriving, which

leads to starvation. Similarly, in SJF long processes may wait longer time if shorter processes keep arriving. Moreover, the algorithm may have poor performance in the worst case scenario. Therefore, it is of a great importance to search for an efficient task scheduling algorithm that distributes the requests to the virtual machines and assures a certain level of quality of service achievement.

This paper proposes a novel task scheduling algorithm called Queue Priority Shortest Job First (QPSJF) Scheduling algorithm. The algorithm is evaluated using the CloudSim simulator. It is based on queues that contain tasks and each task has its priority and length (the number of instructions that a process intends to execute). The QPSJF divides the tasks into three queues according to their lengths and priorities. The first queue is used for the shortest tasks, the second queue for the longest tasks, and the third queue is used for the extremely high priority tasks.

The proposed algorithm maps one task from each queue at the same time to a specific virtual machine. Consequently, minimizing total execution time, makespan, and waiting time. Also, saving energy by mapping all tasks to three virtual machines. Less energy by using a minimal number of virtual machines. Hence, fewer computing resources (e.g. CPU'S, storage). On the other hand, the combination of processing power onto fewer virtual machines allowing results in a higher utilization [3] [14].

The main contribution of this work is develop a novel task-scheduling algorithm for the cloud environment. The proposed algorithm takes in consideration optimality, recourse utilization, and does not suffer starvation.

The rest of the paper is organized as follows: Section 2 walkthrough some related works in task scheduling. Section 3 provides a detailed description of the proposed algorithm. Section 4 discusses the conducted experiments and the obtained results, which are compared with results obtained from other scheduling policies. Finally, Section 5 provides the paper summary along with the conclusion and the suggested future works.

2.0 RELATED WORK

For many decades, the task scheduling problem has been considered one of the very challenging research problems. The search for an efficient solution for the task scheduling problem has been around for a long time and before the emergence of cloud computing (CC). This section outlines some of the previous work related to task scheduling algorithms.

A study presented by N. Panwar in [7] discusses a multi-criteria cloud computing approach. The author proposes a task scheduling algorithm called the TOPSIS-PSO algorithm. The algorithm focuses on the order of processes to provide a solution by operating on two levels. In the first level, the algorithm computes the nearness between the processes according to their execution time, transmission time, and cost. In the second level, the algorithm proceeds to establish a relationship between the processes according to their nearness using Particle Swarm Optimization (PSO). The information obtained from these measures was used for scheduling the tasks. When compared with other task scheduling approaches [7], the proposed algorithm achieved better results. It showed 75% resource utilization improvement and it reduced the computation time by 23.93% and 55.49%. Also, the algorithm reduced the makespan by 29.1%. Unfortunately, the the algorithm does not consider the waiting time of the scheduled tasks.

Hicham Ben Alla et al. [8] presented a hybrid approach based on dynamic dispatch queues. The authors propose two hybrid meta-heuristic algorithms. The first algorithm is hybrid fuzzy logic and the Particle Swarm Optimization (PSO), which is referred to as TSDQ-FLPSO. The second Algorithm is a hybrid Simulated Annealing (SA) and the PSO algorithm, which is referred to as TSDQ-SAPSO. In the TSDQ-SAPSO algorithm, the SA is used to control the inertia weight of a task to minimize its effects on the PSO algorithm, which in turn reduces the probability of a blocked process. The TSDQ-FLPSO algorithm deals with input and output variables to make the scheduling process more accurate and efficient. The two algorithms were compared with each other considering several performance factors. The TSDQ-FLPSO algorithm results in a 1.596 fitness value, 14.09 makespan, 0.92 DI, 82.1% RU, and $281,8 cost. The TSDQ-SAPSO algorithm results in 1.701 fitness value, 14.15 makespan, 0.93 DI, 82 % RU, and $283 cost. The comparative analysis proved the TSDQ-SAPSO algorithm more capable of solving the task scheduling problem. The main limitation of this algorithm was the relatively high waiting time of the scheduled tasks.

X. Geng et al. [11] proposed a task scheduling algorithm in the cloud environment for scientific workflow. The proposed task scheduling algorithm combines task duplication and task grouping, which was referred to as a static DAG scheduling algorithm. The method involves duplicating the joining nodes and converting a DAG (Directed Acyclic Graph). into an in-tree graph. Then, dividing tasks into groups, which reduces communication overhead between tasks. This step is followed by merging some tasks to reduce the load on the processors. Finally, distribute the tasks over the processors based on the idle time of the processors. The analysis showed improvement in the makespan rate, processor utilization, and computation cost. However, due to it is complex computations the execution cost of the algorithm was high.

S. lmougy et al. in [12] proposed a hybrid task scheduling approach that combines the Shortest Job First (SJF) and the Round Robin (RR) algorithms with dynamic variables for quantum time. This hybrid approach works on two levels. The first level contains a variable task quantum that operates dynamically to make the waiting time between short and long tasks as equal as possible. Through the second level, the ready queue is divided into two queues q1 which contains short tasks, and q2 for long tasks. Thus, the algorithm allows two tasks from q1 and one task from q2 to be executed at the same time making the waiting time between q1 and q2 balanced as much as possible. This hybrid approach achieved efficient results in reducing the task waiting time by an average of 18.55 seconds better than the SJF and the RR, in addition to other scheduling algorithms from the literature [12]. However, the algorithm did not consider the makespan time.

S. Banerjee et al., in [13] proposed a task allocation approach based on resource utilization policy. The proposed algorithm partitions the cloudlets into two clusters namely high-end resource cluster (HERC) and low-end resource cluster (LERC) to minimize the execution time for each cloudlet. The partitioning of cloudlets into the two clusters depends on a deadline value. Therefore, the cloudlet is placed into the HERC cluster if its finish time is greater than the deadline value otherwise the cloudlet is placed into the LERC cluster. The proposed algorithm achieved efficient results. It reduced the makespan of HERC and LERC clusters by 9.23 seconds and 30.49 seconds, better than the round-robin and greedy algorithms from the literature [13]. However, the execution cost of the algorithm was high due to it is complex computations.

D. Saxena and R. K. Chauhan [14] proposed an approach that aims to optimize and enhance the task scheduling process by using the SJF algorithm with fair priority and energy realization scheduling. The authors modified the SJF algorithm by using a fair priority policy. The algorithm schedule tasks by sending the maximum possible number of cloudlets to a random VM and then reducing the number of available servers to save more energy. This modification achieved promising results compared when compared with the sequential and the shortest job first algorithms. One concern regarding this approach was that, in some cases, the algorithm scheduled tasks more than ones resulting in high execution time.

H. G. Tani et al. [15] proposed a Smart RR algorithm (SRR), which involves modification to the traditional RR algorithm to enhance the performance of the task scheduling process and satisfy cloud computing and big data needs. SRR operates by adding a smart layer to the current RR algorithm to adapt to every situation in the clousdsim environment. SRR uses a dynamic wait event quantum that will be updated every time there is a task in the waiting queue. The proposed modification achieved efficient results. When it is applied to 10 cloudlets it results in 450 seconds response time and 600 seconds waiting time.

S. Sindhu [16] addressed the task allocation problem by proposing two algorithms to schedule tasks according to their length. The author developed two scheduling algorithms namely the Longest Cloudlet Fastest Processing element (LCFP) and the Shortest Cloudlet Fastest Processing element (SCFP). The LCFP orders the tasks in increasing order according to their lengths then it gives the longest tasks more processing elements to reduce their execution time. On the other hand, the SCFP gives the shortest tasks more processing elements. To compare the performance of LCFP and SCFP, both algorithms were applied on 50 cloudlets. The results showed that the LCFP algorithm outperformed the SCFP algorithm by achieving a 56 makespan rate. Both algorithms produce high waiting times due to the absence of queues to manage the scheduled tasks

F. Ramezani et al. [17] proposed an evolutionary optimization model to be used in the

cloud computing environment for task scheduling. The proposed model combines the Multi-Objective Particle Swarm Optimization (MOPSO) and the Multi-Objective Genetic Algorithm (MOGA). This model operates by minimizing four factors associated with each cloudlet namely the task transfer time, the task execution cost, the task queue length, and the used energy. The experimental results proved the efficiency of the proposed model. When the number of cloudlets is four, the model achieved 260 seconds average transfer time, 56.5% average power consumption, and 4750 seconds execution time. When the number of cloudlets is two, it achieved 260 seconds average transfer time, 71% average power consumption, and 5400 seconds execution time. The algorithm uses too many VM's to schedule tasks which affect the efficiency of the scheduling procedure.

Limbani and Oza [9] proposed a multi-level dynamic scheduling policy that minimizes the load by increasing or decreasing the number of virtual machines. The proposed dynamic approach improves the processing time without considering the cost as a factor that affects the scheduling process. The proposed comparative study in [10] provides very important information about several significant service routing policies. The study emphasizes the importance of fully optimizing the service routing policy to keep count of all the factors that affect the scheduling process. Therefore, improving the performance of the scheduling process while reducing the cost [10].

Z. Zhou et al. in [18] proposed a heuristic named MGGS that combines a modified GA algorithm with a greedy strategy to optimize and enhance the task scheduling process. The authors claimed that the optimal solution to schedule all tasks were achieved using few iterations. similarly, many studies proposed Heuristic approaches for improving task scheduling [19][20][21][22][23]. The drawback of the heuristic approach in general, is the high probability of generating high execution time due to their complex natures.

In this section, we went through some of the proposed methods for task scheduling algorithms. Each of these algorithms suffers some drawbacks such as the computational time and the delay. In addition to other limitations that are related to the SJF and priority scheduling. Therefore, the need for developing new techniques to overcome the limitations in the recent technique is vital.

3.0 METHODOLOGY

This section provides detailed information about the proposed QPSJF algorithm by addressing and discussing all the algorithm steps, identifying the relationships between them, and providing a detailed workflow that describes how the algorithm operates. We provide a brief description of the shortest job first (SJF) algorithm, then we discuss the implementation environment, which is followed by describing the QPSJF workflow.

3.1 SHORTEST JOB FIRST ALGORITHM

The shortest job first algorithm is one of the most commonly used task scheduling algorithms in many applications [14]. The SJF algorithm works by sorting the tasks in ascending order from the shortest task to the longest task according to their expected execution time. Thus, each time scheduling is performed; the shortest task will be selected and executed next until no tasks are left. The most significant advantage of this algorithm is that it reduces the average waiting time among all the tasks. The main concern regarding the SJF algorithm is the need to know the execution time for each task beforehand and this is almost impossible in many environments. On the other hand, starvation is considered a major issue, when the shortest tasks keep executes first, the longer jobs may never execute if shorter tasks keep arriving.

The proposed approach involves sorting all tasks in ascending order according to their lengths. Then, distribute these tasks over three queues, the distribution process takes into consideration two factors, the length and the priority of the task. Whenever a mapping between the cloudlets and the VM's occurs, three tasks will be sent at the same time (the shortest one from each queue) to their specific VM's to be executed. The described above-integrated workflow of the QPSJF algorithm solves the starvation problem. Therefore, the QPSJF is can be proven to be an efficient and powerful enhancement to the current SJF algorithm.

3.2. IMPLEMENTATION ENVIRONMENT

The behavioral implementation of the QPSJF algorithm has been simulated on the CloudSim toolkit 3.0.3 simulator, which provides an integrated and powerful environment to simulate many algorithms including task scheduling algorithms. The simulation environment is

configured over the X86 system architecture running on the Linux operating system.

The simulation parameters consist of one broker, two datacentres, 2 hosts with 2 Processing Elements (PE), 3 virtual machines, and 40 cloudlets. At each run, the lengths and the priorities of all cloudlets are randomly generated from 1000 – 3000 instructions and from 1 – 70 priorities to generate more realistic task execution. Table 1 shows the parameters for the two hosts and Table 2 shows the parameters for the VMs.

Table 1. Hosts Parameters.

Host ID	RAM	Storage	Bandwidth	Number of PE	Utilization Mode
0	6 GB	20000	20000 MB/S	2 PE	Full Utilization
1	6 GB	20000	20000 MB/S	2 PE	Full Utilization

Table 2. Virtual Machines Parameters.

VM ID	RAM	Storage	Bandwidth	MIPS	Number of PE	Name	Scheduling Mode
0	1 GB	10000 MB	1000 MB/S	5000 MIPS	2PE	Xen	Space Shared
1	6 GB	10000 MB	1000 MB/S	1000 MIPS	2PE	Xen	Space Shared
2	1 GB	10000 MB	1000 MB/S	1000 MIPS	6 PE	Xen	Space Shared

3.3 QPSJF ALGORITHM WORKFLOW

The proposed algorithm workflow consists of seven steps. These steps are portrayed in the flowchart shown in Figure 1.

The workflow of the QPSJF Algorithm is as follows:

1. Sorting the step cloudlets: after all cloudlets are created, they are sorted in ascending order according to their lengths (number of instructions) to produce a sorted list that contains all of the cloudlets. This step results in minimizing the execution time for all cloudlets.
2. Computing the Average Length AL of all cloudlets and the priority threshold T: let $C = \{c1, c2, c3 \dots cn\}$ be the set of all cloudlets that will be executed, let $L = \{l1, l2, l3 \dots ln\}$ be the set of all cloudlets lengths and let $P = \{p1, p2, p3 \dots pn\}$ be the set of all priorities among all cloudlets. Thus, the AL is the average length of the cloudlets in list L.

$$AL = \frac{(l1+l2+l3+\cdots+ln)}{n} \quad (1)$$

The value of T for all cloudlets is the difference between lowest priority (LP) and highest priority (HP) divided by 2.

$$T = \frac{HP-LP}{2} \quad (2)$$

The T value is used to distinguish the high priority cloudlets from the low priority ones. The value of T is computed as a median value because the priorities among the cloudlets are randomly generated. Thus, T will guarantee that all cloudlets are covered and distinguished into high priorities cloudlets and low priorities cloudlets. The AL value distinguishes the cloudlets with long lengths from those with short lengths.

3. Queues creation: this step involves creating and initializing three queues to store the cloudlets according to their lengths and priorities.
4. Cloudlets allocation: this step allocates each cloudlet to the destination queue. based on the resulted values of AL and T the cloudlets will be divided into three queues namely the Shortest Cloudlets Queue (SCQ), the Longest Cloudlets Queue (LCQ), and the Highest Priority Queue (HPQ). The SCQ contains all of the cloudlets that have short lengths and low priorities, the LCQ contains all cloudlets having long lengths and high priorities and the HPQ contains all cloudlets having extremely high priorities.

For example, given a cloudlet ci with a length li and a priority value pi. To place the cloudlet ci in the proper queue, we test the value of the corresponding li and pi against the average length AL and the threshold T using the following scenario:

$$if(li < AL) and (pi < T) \rightarrow insert(SCQ, ci)$$

$$if(li \geq AL) and (pi \geq T) \rightarrow insert(LSQ, ci)$$

$$if(pi > T) \rightarrow insert(HPQ, ci)$$

It is worth mentioning that the cloudlets in the SCQ and LCQ are sorted in ascending order

according to their lengths, and the cloudlets in HPQ are sorted in ascending order according to their priorities.

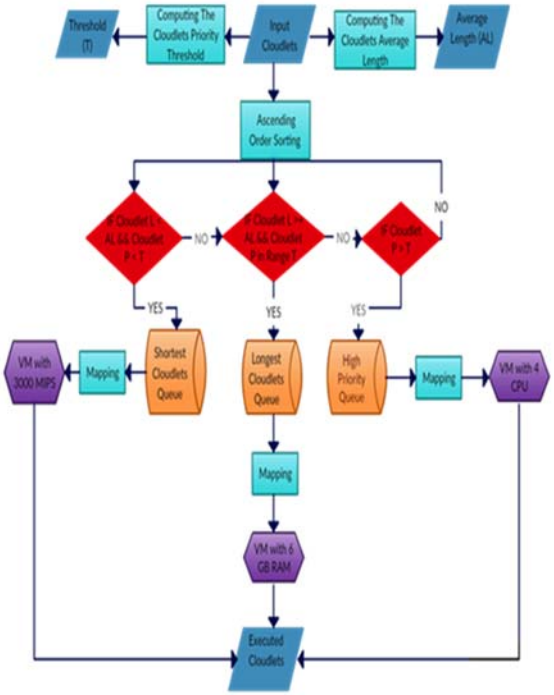

Figure 1. QPSJF Algorithm Workflow

5. cloudlets - VM mapping: having three queues filled with cloudlets according to the previously described allocation, the cloudlets are mapped to VM's as follows: all the SCQ cloudlets are mapped to the VM that has 6 GB RAM, the LCQ cloudlets are mapped to the VM that has 5000 MIPS processing power, and the HPQ are mapped to the VM that has 6 processing elements (processors). This step aims at minimizing the execution time for each cloudlet.
6. Task execution: to minimize the waiting time, and the response time for each cloudlet, three cloudlets are sent at the same time (the shortest one from SCQ, the shortest one from the LCQ, and the lowest priority cloudlet from the HPQ) to the VM's. In case if one of the virtual machines is busy executing other cloudlets, the arrived cloudlet will be sent to the next available virtual machine. Thus, this will ensure that the waiting time is reduced to its possible minimum value.
7. Output generation: this step represents all the scheduled cloudlets according to their lengths and priority and gives the information about each cloudlet including execution time, waiting time, cloudlet length, and cloudlet priority.

3.4 QPSJF PSEUDOCODE

Figure 2 describes the scheduling operation of the proposed QPSJF algorithm.

```
1:  begin
2:     create Virtual Machines VM = {vm1, vm2, vm3}
3:     create Cloudelts C = {c1, c2, ... ci}
4:      create Queues Q = {SCQ, LCQ, HPQ}
5:     sort C in ascending order according to their lengths and priorities to obtains Sorted List LS
5:     compute AL=(l1+l2 +l3+?+ln)/n
6:     compute T=(HP-LP)/2

7:      for(LS:C)
8:            if(ci.l < AL && ci.pi < T)
9:               SCQ.ADD(ci)
10:           else if(ci.l >= AL && ci.pi > T && ci.p <= T + 13)
11:           LCQ.ADD(ci)
12:           else if(ci.p >= T + 13)
13:              HPQ.ADD(ci)
14:      end

15:      while(i != C.size)

16:          if(i <= SCQ.size)
17:               map SCQ.ci to vm1
18:           else if(i <= LCQ.size)
19:               map LCQ.ci to vm2
20:           else if(i<= HPQ.size)
21:               map HPQ.ci to vm3
22:           else if(i == C.size)
23:               print("all cloudets finshed mapping")
24:      end

25:     print cloudlet information

26:     end
```

Figure 2: QPSJF Pseudocode describing how the proposed algorithm schedules tasks

The pseudocode in Figure 2 shows the main steps of the QPSJF algorithm, which involves creating the three main queues and sorting the tasks in a non-decreasing order of task length. Then, distribute the sorted tasks over the three queues according to their length and assigned priority. Consequently, three cloudlets at a time will be transferred to the designated virtual machine whenever mapping occurs.

In this section, we presented the detailed steps of the proposed scheduling algorithm. The algorithm starts by sorting the cloudlets in ascending order of lengths, which guarantees optimality in scheduling by reducing the average waiting time. Then, the algorithm calculates the priority of each cloudlets. After that, the cloudlets are grouped into three main categories depending on the combination of the length and the priority of each cloudlets. Each

of the generated groups forms a priority queue of cloudlets that combines properties of length and priority. The combination of length and priority is selected carefully to insure optimality. On the other hand, the proposed selection avoids starvation by assuring the fair execution of all processes including the long ones and the low priority ones. Finally, each of the three queues is mapped to a specific virtual machine the suites the process execution requirements.

4.0 EXPERIMENTAL RESULTS

To evaluate the performance of the proposed QPSJF algorithm and to compare the obtained results with other scheduling algorithms, an experimental environment was created consisting of forty cloudlets, three VM's, two hosts, two datacentres, and one service broker. The main parameters that were applied to measure the efficiency of the proposed approach are execution time, waiting time, response time

The following three scheduling parameters are used as metrics to evaluate the performance of the proposed QPSJF task scheduling algorithm.

- ✓ Average Waiting Time AWT. The waiting time for task i is the amount of time in seconds that the cloudlet spends in the waiting queue.

$$AWT = \frac{\sum WT_i}{n} \quad (3)$$

- ✓ Average Execution Time AXT. The execution time for task i is the amount of time in seconds that the cloudlet spends in the virtual machine.

$$AXT = \frac{\sum XT_i}{n} \quad (4)$$

- ✓ Average Response Time ART. The response time for task i is the amount of time in seconds that the cloudlet spends before it starts execution.

$$ART = \frac{\sum RT_i}{n} \quad (5)$$

The results obtained showed a significant improvement in the values of these parameters. The results are illustrated in Table 3 through table 6.

Table 3. The QPSJF Algorithm Results

Number of Cloudlets	Execution Time (seconds)	Response Time (seconds)	Wating Time (seconds)
10	6.5	90	35
20	20	130	70
40	39	185	142
50	78	215	250
70	166	350	340
100	342	440	435
200	600	520	553

Table 3 shows the performance results of the proposed QPSJF algorithm compared to other algorithms from the literature. The proposed QPSJF algorithm achieved efficient results. The performance of the QPSJF algorithm has been evaluated using several cloudlets dataset, starting from a dataset that contains 10 cloudlets and finishing with a dataset that contains 200 cloudlets. Therefore, when we used 10 cloudlets to test the performance of the QPSJF algorithm, it achieved 6.5 seconds execution time by an average of 0.65 seconds, 35 seconds waiting time by an average of 3.5 seconds. When we used 50 cloudlets to test the performance of the proposed algorithm, it achieved 78 seconds execution time by an average of 1.56 seconds, and 280 seconds waiting time by an average of 5.6 seconds. Also, when we used 200 cloudlets to test the performance of the QPSJF algorithm it achieved 600 seconds execution time by an average of 3 seconds, and 553 seconds waiting time by an average of 2.75 seconds.

Table 4. The execution time in seconds of QPSJF compared with other algorithms

Number of Cloudlets	QPSJF	SJF with fair priority	FCFS	Min-Min	MGGS
20	20	25	130	200	170
40	39	130	224	260	230
50	78	--	400	330	300
70	166	--	650	480	430

The execution performance of the QPSJF algorithm is presented in table 4 and figure 3, the performance of the proposed QPSJF algorithm has been evaluated by comparing it with the algorithms (SJF with fair priority, FCFS, Min-Min, and MGGS) in [14, 18]. Therefore, the results in table 4 and

figure 3 show that the proposed QPSJF algorithm outperforms the compared algorithms in minimizing the execution time. The missing values (--) in the SJF with fair priority algorithm column indicate that the algorithm was not tested using 50 and 70 cloudlets. Also, in the proposed experiments, the QPSJF algorithm was tested using 3 virtual machines, on the other hand, the SJF with fair priority algorithm was tested using 6 virtual machines in [14], and both Min-Min and MGGS algorithms were tested using 10 virtual machines in [18].

Table 5. The total response time of QPSJF compared with other algorithms.

Number of Cloudlets	QPSJF	FCFS	Min-Min	MGGS
20	130	210	160	180
40	185	250	210	220
50	215	290	240	250
70	350	480	380	400

The response performance of the QPSJF algorithm is presented in table 5 and figure. The proposed QPSJF algorithm has been compared with the algorithms (FCFS, Min-Min, and MGGS) in [14, 18]. The results in table 5 and figure 4 show that the proposed QPSJF algorithm outperforms the other algorithms in minimizing the response time. In the proposed experiments, the QPSJF algorithm was tested using three virtual machines, and both Min-Min and MGGS algorithms were tested using 10 virtual machines in [18].

Table 6. The waiting time in seconds of our proposed QPSJF compared with other algorithms

Number of Cloudlets	QPSJF	Smart Round Robin	SJF
10	35	250	25
40	136	-	200

Table 6 and Figure 5 describe the performance of the QPSJF algorithm compared with the benchmark algorithms (SJF, and smart round-robin algorithms) in [15] for the average waiting time. Therefore, the results in table 5 and figure 4 show that the proposed QPSJF algorithm outperforms the compared algorithms in minimizing the waiting time for all cloudlets. The missing value (--) in the smart round-robin algorithm column indicates that the algorithm was not tested using 50 and 70 cloudlets. Besides, in the proposed experiments, the QPSJF algorithm was tested using three virtual machines; on the other hand, the SJF and the smart round-robin algorithms were tested using six virtual machines in [15].

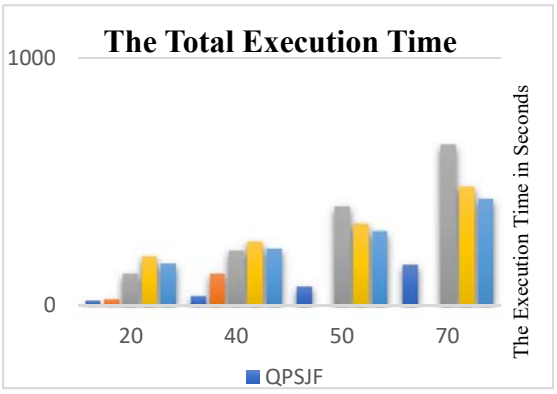

Figure 3. The execution time of QPSJF compared with other algorithms.

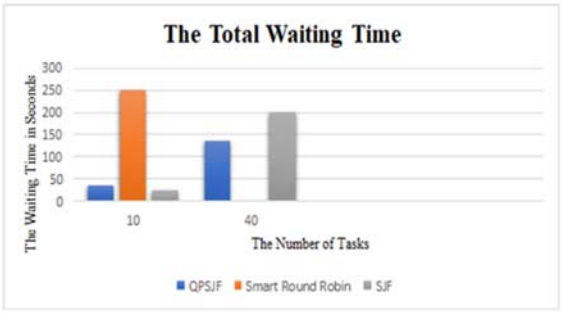

Figure 4. The total response time of QPSJF compared with other algorithms.

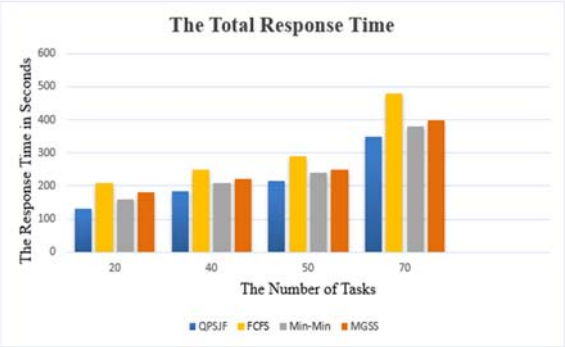

Figure 5. The waiting time QPSJF compared with other algorithms

According to the results presented in Figure 3 through Figure 5, and Table 4 through Table 6, the QPSJF algorithm enhanced the performance of the task scheduling process. Thus, it minimized the execution time, waiting time, and response time comparing to other prosed benchmark algorithms

In this section, we tested the performance of the proposed scheduling algorithm (QSJF) and compared the results to other proposed scheduling algorithm from the letreture. The main parameters that were applied to measure the efficiency of the proposed approach are makespan, execution time, waiting time, response time.

5.0 CONCLUTION

This paper proposes a novel task scheduling approach (QPSJF) that uses multiple queues and combines properties of different classical scheduling algorithms to arrange the tasks to be mapped to the virtual machines efficiently.

The algorithm sorts the cloudlets in increasing order of the task length, which achieves optimality in reducing the total execution time, waiting time, and response time.

Based on the cloudlet's priority and a threshold value, the cloudlets are distributed among three main priority queues (namely, the shortest cloudlets queue, the longest cloudlets queue, and the highest priority queue). This process helps accelerate the execution of the higher priority cloudlets without causing starvation to the other low priority cloudlets.

Each of the three generated queues is mapped to a specific virtual machine so that whenever mapping occurs, three cloudlets are sent at the same time (the shortest one from SCQ, the shortest one from the LCQ, and the highest priority cloudlet from the HPQ) to their specific VM's to be executed. This process helps in maintaining energy while enhancing recourse utilization.

The simulation results show that the QPSJF algorithm enhanced the performance of the task scheduling process by minimizing the average waiting time, the average execution time, and the average response time. On the other hand, QPSJF algorithm helps enhance resource utilization while maintaining energy. In the future, we will continue our study by implementing the algorithm in real-life cloud computing environments.

REFERENCES:

[1] M. R. Rahimi, J. Ren, C. Harold, A. V Vasilakos, and N. Venkatasubramanian, "Mobile Cloud Computing : A Survey, State of Art and Future Directions," *Mob. Networks Appl.*, vol. 19, no. 2, pp. 133–143, 2014.

[2] Zhang, Q., L. Cheng, and R. Boutaba, Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications, 2010. 1(1): p. 7-18.

[3] S. Singh and I. Chana, "A Survey on Resource Scheduling in Cloud Computing ," *J. Grid Comput.*, vol. 14, no. 2, pp. 217–264, 2016.

[4] R. Sosan and C. F. Azim, "RETRACTED ARTICLE : Mobile Cloud Computing : The Taxonomy and Comparison of Mobile Cloud," *Wirel. Pers. Commun.*, vol. 89, no. 4, p. 1435, 2016.

[5] A. Siavashi and M. Momtazpour, "GPUCloudSim : an extension of CloudSim for modeling and simulation of GPUs in cloud data centers," *J. Supercomput.*, vol. 75, no. 5, pp. 2535–2561, 2019.

[6] M. Shiraz, S. Abolfazli, Z. Sanaei, and A. Gani, "A study on virtual machine deployment for application outsourcing in mobile cloud computing," *J. Supercomput.*, vol. 63, no. 3, pp. 946–964, 2013.

[7] N. Panwar, "TOPSIS – PSO inspired non-preemptive tasks scheduling algorithm in cloud environment," *Cluster Comput.*, pp. 1–18, 2019.

[8] H. Ben Alla, S. Ben Alla, and H. Ben Alla, "A novel task scheduling approach based on dynamic queues and hybrid meta-heuristic algorithms for cloud computing environment," *Cluster Comput.*, vol. 21, no. 4, pp. 1797–1820, 2018.

[9] Limbani, D. and B. Oza, A Proposed Service Broker Policy for Data Center Selection in Cloud Environment with Implementation. International Journal of Computer Technology & Applications, 2012. 3(3).

[10] Mishra, R.K., S. Kumar, and B. Sreenu Naik. Priority based Round-Robin service broker algorithm for Cloud-Analyst. in Advance Computing Conference (IACC), 2014 IEEE International. 2014.

[11] X. Geng, Y. Mao, M. Xiong, and Y. Liu, "An improved task scheduling algorithm for scientific workflow in cloud computing environment," *Cluster Comput.*, pp. 1–10, 2018.

[12] S. Elmougy, S. Sarhan, and M. Joundy, "A novel hybrid of Shortest job first and round Robin with dynamic variable quantum time task scheduling technique," *J. Cloud Comput.*, 2017.

[13] S. Banerjee, A. Roy, A. Chowdhury, R. Mutsuddy, R. Mandal, and U. Biswas, "An Approach Toward Amelioration of a New Cloudlet Allocation Strategy Using Cloudsim," *Arab. J. Sci. Eng.*, vol. 43, no. 2, pp. 879–902, 2017.

[14] D. Saxena and R. K. Chauhan, "Shortest-Job First With Fair Priority and Energy Awareness Scheduling In Green Cloud Computing," *Int. J. Trend Res. Dev.*, vol. 3, no. 6, pp. 373–377, 2016.

[15] H. G. Tani *et al.*, "Smarter Round Robin Scheduling Algorithm for Cloud Computing and Big Data To cite this version : HAL Id : hal-01443713 Smarter Round Robin Scheduling Algorithm for Cloud Computing and Big Data," *J. Data Min. Digit. Humanit. Episciences.org*, 2018.

[16] S. Sindhu, "Efficient Task Scheduling Algorithms for Cloud Computing Environment Efficient Task Scheduling Algorithms for Cloud Computing Environment," in *High Performance Architecture and Grid Computing*, 2016, no. January 2011.

[17] F. Ramezani, J. Lu, J. Taheri, and F. K. Hussain, "Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments," *World Wide Web*, vol. 18, no. 6, pp. 1737–1757, 2015.

[18] Z. Zhou, F. Li, H. Zhu, H. Xie, J. H. Abawajy, and M. U. Chowdhury, "An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments," Neural Comput. Appl., vol. 31, pp. 1–11, 2019.

[19] I.Strumberger, N. Bacanin, M. Tuba, E. Tuba, Resource Scheduling in Cloud Computing Based on a Hybridized Whale Optimization Algorithm, Applied Sciences, Vol. 9, No. 22, pp. 4893 - 4893, Nov, 2019.

[20] Kalra, M.; Singh, S. A review of metaheuristic scheduling techniques in cloud computing.Egyptian Informatics Journal 2015, 16, 275 – 295.

[21] I. Strumberger, M. Tuba, N. Bacanin, E. Tuba, Cloudlet Scheduling by Hybridized Monarch Butterfly Optimization Algorithm, Journal of Sensor and Actuator Networks, Vol. 8, No. 3, pp. 44 - 44, Aug, 2019.

[22] Ameera Jaradat, "Rational graph: a model for complex networks" , International Journal of Web Engineering and Technology 13 (1), 56-77, (2018).

[23] Sreenu, K.; Sreelatha, M. W-Scheduler: whale optimization for task scheduling in cloud computing. Cluster Computing 2017.

[24] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: a literature survey," Future Generation Computer Systems, vol. 91, pp. 407–415, 2019.