An Efficient Task Scheduling Algorithm for Private Cloud Computing based on User priority

خوارزمية فعاله لجدولة المهام في الحوسبة السحابية الخاصة مبنية على أولوية المستخدم

By

Mohammad Al-Zyadneh

Supervisor

Prof. Ismail Ababneh

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree of Science in Computer Science

Deanship of Graduate Studies

Al al-Bayt University

May, 2019

Committee Decision

This Thesis (An Efficient Task Scheduling Algorithm for Private Cloud Computing based on User priority was successfully defended and approved on 00/05/2019.

Examination Committee	Signature
Prof. Ismail Ababneh (Supervisor) ……..……	
Prof. Saad Bani-Mohammad ……..……	
Prof. Omar Shatnawi ……..……	
Dr. Wail Elias Mardini ……..……	

Dedication

This thesis is dedicated to

My parents along with all my family members

For their love, endless support and encouragement.

Acknowledgments

First of all, I would like to express my deep gratitude to my supervisor, Dr. Ismail Ababneh for his inspiring guidance, valuable advice, and constant encouragement throughout the progress of this work. His suggestion and his frequent questions motivated this thesis and he never failed to provide his help at all stages of this thesis.

My great thanks are for my parents, my brother, and my sisters, without their encouragements and support I could not do anything.

Appreciate all of my friends who encouraged me during my master study; they truly helped me a lot.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

An Efficient Task Scheduling algorithm for Private Cloud Computing based on User priority

By

Mohammad Al-Zyadneh

Supervisor

Prof. Ismail Ababneh

Abstract

Nowadays, Cloud computing has gained much attention in many applications. The user can use cloud resources on demand on a pay-as-you-go from anywhere and at any time. The cloud computing environment is suitable for serving a large number of tasks using the available computing resources. The scheduling algorithm is an important factor in cloud computing environment as it manages the order of execution of the tasks with the goal of improving the throughput of the cloud computing resources. In private cloud computing, user priority is one of the major user needs inside the organization that should be taken into account, where priority is given to user tasks that should not be late. However, most researchers have not attempted to solve the starvation problem that can occur in priority-based systems.

In this thesis, an efficient task scheduling method named High-Priority-Job-First (HPJF) that is based on user priority is proposed for private cloud computing systems. The suggested method assigns tasks to cloud resources in an efficient manner based on user type, execution cost, task execution time, and load on the virtual machine. In addition, a multi-queue technique is used to overcome the problem of starvation that occurs in priority-based systems. HPJF is Implemented using a simulation called CloudSim. HPJF was compared with four scheduling algorithms First Come First Serve (FCFS), Round Robin (RR), Short Job First (SJF) and Best Level Job First (BLJF). The simulation results show that HPJF has better performance in terms of both waiting time and turnaround time compared with the other tasks scheduling algorithms.

Chapter 1
Introduction

1.1 Overview

Recently, cloud computing has received the attention of organizations and users due to the high performance computing services and facilities that it provides to end users. Cloud computing is defined by the National Institute of Standards and Technology as "A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources such as networks, servers, storage, applications, and services that can be rapidly provisioned and released with minimal management effort or service provider interaction" (Wyld, 2009). Cloud computing is also defined in terms of the two words cloud and computing. The term "Cloud" refers to the applications that provide services to end users and the hardware and system software that are responsible for providing the services. While the term "Computing" refers to carrying out user tasks with higher resources availability and lower cost (Katyal & Mishra, 2014).

There are three main stakeholders of clouds represented by end users, cloud providers and cloud developers. The customers that use the various services (infrastructure/software/platform) of the cloud are known as the **end users**. The users request the various services after adhering to the Service Level Agreement (Allan et al.) by the Cloud Provider on a pay-per-use model (Aslanzadeh, 2016). The user's request (known as **task**) is a basic unit of user request it denotes an independent unit of computation, (naturally a program and possibly associated data) to perform on a machine. Each task may have a certain priority level, an earliest possible starting time and a due date (Baeza-Yates & Ribeiro-Neto, 1999). Whereas, a machine, also known as resource, is a basic unit of scheduling and each machine has its own elements such as CPU, memory, system model, operating system, software, etc. The cloud developer is responsible for meeting the requirements of both the cloud user and the cloud provider. Provider offers

four different infrastructures and manages their resources to be provided to the end users as illustrated in Figure 1.1 (Katyal & Mishra, 2014).

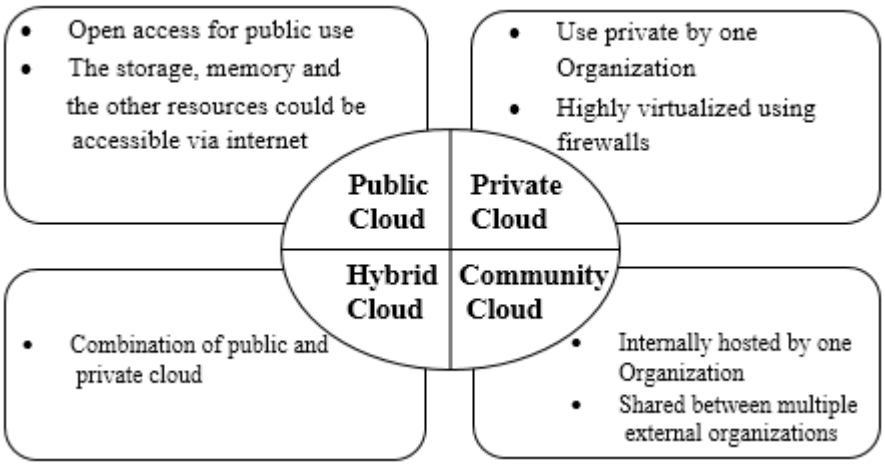


Figure 1.1: Cloud infrastructures (Katyal & Mishra, 2014)

1. **Public cloud:** The cloud infrastructure in this type of cloud is available for public usage. It may be owned, managed, and operated by a business, government, academic or organization, or any combination of them.
2. **Community cloud**: This type of cloud can be used exclusively by a specific organization, community and also the consumers who have sharing objectives (e.g., security mission). The cloud also can be owned, managed, and operated by one or more of the organizations in the community, a third party, or any combination of them.
3. **Private cloud**: This type of cloud is the main focus of this thesis where it refers to cloud computing on private networks in which cloud can be used exclusively by a single organization, including several customers (e.g., business, company…). It can be owned, operated, and managed by the organization itself, a third party, or a combination of both of them.
4. **Hybrid cloud**: The cloud infrastructure in this type of cloud is a combination of two or more cloud infrastructures (private, community, or public) (S. Kaisler, W. H. Money & S. J. Cohen, 2012). In the hybrid cloud, a standardized or proprietary

5. technology is used to allow organization can manage some resources internally and externally.

1.2 Benefits and Drawbacks of the Private Cloud

There are many features and benefits for using private cloud computing such as high security, privacy, more control, cost and energy efficiency, improved reliability and cloud bursting (Ghazizadeh, 2012) as follow:

1. Higher Security and Privacy

Private clouds use techniques such as distinct pools of resources with the restrictive access made from the application of firewalls, dedicated leased lines and internal hosting for protecting the system from hackers Also, typically no user can see all information in such system.

2. Better control

A private cloud is only accessible by a single organization, that organization will have the ability to configure and manage the implementation and the infrastructure; servers, firewalls, networks and communication, middleware, Also, it has control over the security implementation.

3. Improved reliability

In a private cloud, resources (servers, networks etc.) are hosted internally, and the creation of virtualized operating environments reduces the individual failures in the resources. Therefore, private clouds make resources more resilient across the physical infrastructure.

4. Cloud bursting

One of the features for private cloud is cloud bursting. The providers may offer the opportunity to employ cloud bursting when the demand for computing capacity spikes. This service allows an application to run in a private cloud and to burst into a public

cloud to free up more space in Private cloud for the sensitive functions that require it.

5. Cost and energy efficiency

Implementing a private cloud model can improve the allocation of resources within an organization by ensuring the availability of resources to individual departments/business functions and responding to their demand directly and easily. They make more efficient use of the computing resource than traditional LANs, this allows *reducing* energy consumption.

- **Drawbacks of Private cloud**

There are mainly three drawbacks of private clouds (Zhang, Cheng, & Boutaba, 2010) as follow:

1. Higher cost

The major drawback of the private cloud is its higher cost compared to the public cloud; the cost of purchasing equipment, software and staffing often results in higher costs to an organization having their own private cloud.

2. Scalability

The private cloud is not easy to scale compared to the public cloud, because adding physical hardware require security restrictions, and may require the organization to rent out more space or even move locations to have enough space to allocate all the required hardware.

3. More maintenance

In private clouds, organizations have to take care of all hardware and underlying networks; they require daily and weekly maintenance If not properly maintained, the organization runs the risk of losing data.

1.3 Service Models:

The services in cloud computing are categorized into three major approaches: Infrastructure as a Service (**IaaS**), Platform as a Service (**PaaS**), and Software as a Service (**SaaS**), **IaaS** provides infrastructure service in virtual platform for the user to achieve various virtual kinds of work such as processing, storage, and server functions, in which the users purchase the services similar to electricity based usage, **PaaS** provides platform service for users to allow them to develop the applications and to check the output quickly and effectively, **SaaS** provides the software to the user, application can be run directly without the need to install run applications on a personal computers (Mell & Grance, 2011).

The cloud providers have huge computing resources in their datacenters (DCs), which can be rented to the end users per-usage. In contrast, users run the applications on the cloud pay for the provider the required cost according to applications loads and the resources usage. The scenario starts from the users to the Service Broker. The Service Broker selects a suitable datacenter according to the service broker policy (Guo, Zhao, Shen, & Jiang, 2012). DCs consist of a number of physical hosts that manages a number of Virtual that manage (VMs) as portrayed in Figure1.2 (Mell & Grance, 2010)

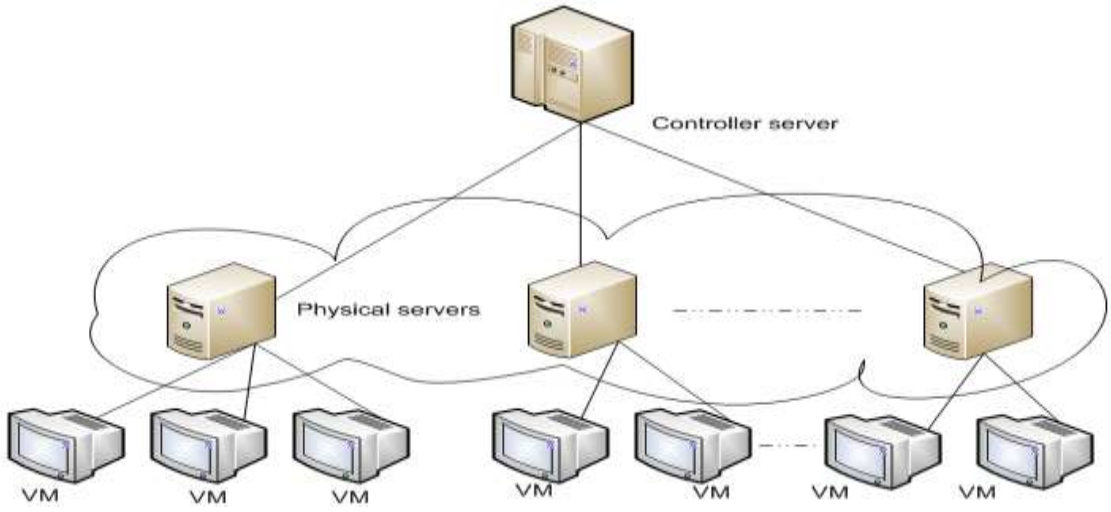


Figure 1.2: Cloud computing architecture (Mell & Grance, 2010).

Several techniques are used to serve all user requests without long waiting time. Task scheduling is one of the techniques that is used to improve the performance of cloud computing system. In the next section, task scheduling is discussed.

1.4 Task scheduling

Task scheduling is defined as the process that specifies the order of selecting a job located within the waiting queue for processing (Babbar and Krueger, 1994; Ababneh and Bani-Mohammad 2011). Task scheduling is used in cloud computing to improve the performance of the system (Khajemohammadi, Fanian, & Gulliver, 2013). Task scheduling can be done in two modes; space shared and time-shared. In space-shared mode, the resources are not preempted until the completion of the tasks execution. While in the time-shared mode, resources are preempted until all the tasks complete their execution. There are two types of tasks scheduling based on scheduling decisions; the static and the dynamic Scheduling. In static scheduling, the scheduling decisions are based on the parameters fixed before submitting the tasks for execution (Li, 2012). On the other hand, the dynamic scheduling decisions are based on dynamic parameters that may change during the execution of the tasks (Hu, Gu, Sun, & Zhao, 2010).

There are two types of tasks that can be scheduled; the independent tasks where each task is **i**ndependent form the other tasks and scheduled without considering their interdependence. Dependent tasks that are known as workflow tasks, where there are dependencies between tasks that should be taken into account when assigning the tasks to the available resources (Aslanzadeh, 2016).

Different scheduling algorithms are used to manage the execution of the user's tasks which have different properties such as Shortest-Job-First (SJF), Round Robin(RR), First-Come-First-Serve (FCFS) and Multilevel queue scheduling (MQ) as follow:

1.4.1 Shortest-Job-First (SJF)

Shortest job first (SJF) is a scheduling algorithm that selects the task with the smallest execution time from the waiting queue to be executed next. Shortest Job first scheduling

algorithm is easy to implement but it may suffer from starvation problem. However, this problem can be resolved by the aging concept (Ru & Keung, 2013).

.1.4.2Round Robin (RR)

Round Robin (RR) is simple, fair and widely used, where the tasks in a queue are served in circular order and each task equal chance (time slice) to get a resource. Round Robin scheduling is easy to implement, and starvation-free. (Sotomayor, Montero, Llorente, & Foster, 2009).

1.4.3 First-Come-First-Serve (FCFS)

First-Come-First-Serve (FCFS) executes tasks in the order of their arrival, where new tasks are placed at the end of the waiting queue and wait for the previous tasks to be executed. The First-Come-First-Serve scheduling algorithm is easy to implement and starvation-free (Agarwal & Jain, 2014).

1.4.4 Multilevel queue scheduling (MQ)

Multilevel queue scheduling (MQ), which is the main focus of this thesis, classifies tasks into different priority groups (Agarwal & Jain, 2014). For example, a multilevel queue that has three queues is shown in Figure 1.3.

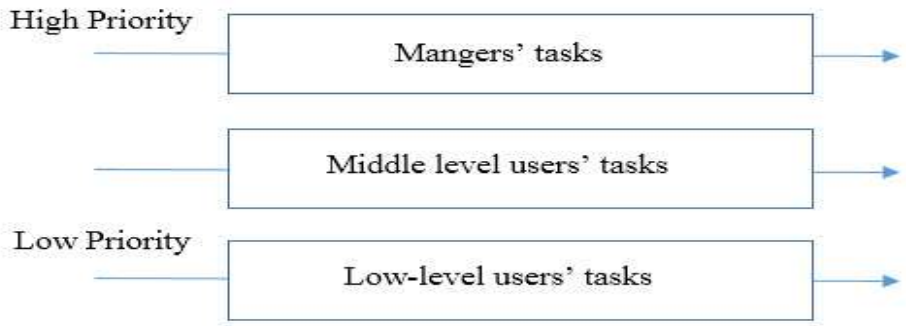


Figure 1.3: A multilevel queue

Each queue has its priority compared with the other queues as can be shown in figure 1.3.

MQ has high performance as compared to the other scheduling algorithms for the cloud environment (Kumaresh et al, 2012;Agarwal & Jain, 2014).

1.6 Problem Statement

One of the most important problems in cloud computing is tasks scheduling. In scheduling process, some tasks may execute before others which can increase the average waiting and turnaround time relatively for low priority tasks when the priority are given for high user to perform their tasks. The classic scheduling algorithms such as SJF, RR and FCFS don't take any consideration for the user's needs. Therefore, the scheduling process needs more scheduling policies to improve the performance of the system.

In addition, the starvation problem is one of the major challenges that priority-based task scheduling suffers from. In private cloud computing, lower priority tasks can starve because execution priority is given to the higher priority tasks. A task may wait for a very long time to be executed. In this research, an efficient scheduling method is proposed to solve the starvation problem of user tasks in the cloud environment. The suggested method uses a multi-queue technique to distinguish the priority of the user tasks based on the user type, task execution cost, task execution time and load on the VM. HPJF uses the multi queue technique to solve the starvation problem that may face the lower priority because execution priority is given to the higher priority tasks.

1.7 Research Objectives

The main objective of the proposed method is to solve the scheduling problem in private cloud computing for the organizations that are scheduling their tasks based on user priority. The following objectives have been delineated:

1. Reduce the overall waiting time for scheduling tasks in cloud environment.
2. Distributing user tasks on the hierarchical queue system and arrange the tasks within each queue according to their task attributes.

3. Give low priority tasks a chance to be scheduled with the high level tasks to avoid starvation problem.

1.8 Research Questions

In order to reach the objectives stated previously, the following questions have to be answered in this thesis:

1. Can an efficient scheduling method for scheduling problem reduce waiting time when the lower level user's tasks are increased in the organization?
2. Can an efficient scheduling method distribute users' tasks on the hierarchy queue system and arrange the tasks within each queue according to their tasks attributes?
3. Can an efficient scheduling method use multi queue technique to avoid starvation problem?

1.9 Scope of Research

The scope of this study concerns primarily on tasks scheduling problem in the cloud environment. The proposed method used two techniques; sort tasks based on the task weight to improve the quality of service (QoS) and the multi queue technique is used to solve the starvation problem that may face the low priority tasks when the execution priority is given to the higher priority tasks. In this research, The tasks weight is represented by four attributes; the user priority, the execution time, the execution cost and the system load the cloud system load and the user priority level. The execution time that represents the time needed to execute the task on the resource. The execution cost denotes the price required to use the resource. The VM load represents the load in the private cloud resources. The user priority level declares the user position in the organization.

1.10 Research Significance

The main significant of this research is the using of an efficient method for solving the scheduling problem. The proposed method is expected to give better results in distributing the tasks over the VM, because it combines the advantages of the two powerful techniques; the Multi-queue technique and the priority-based scheduling

algorithm. The advantages of the Multi-queue technique are represented by given the priority to some tasks than the other and to overcome the problem of starvation. Whereas, the advantages of the priority-based scheduling algorithm is represented by improve quality of service (QoS) for all users where each user is given an appropriate QoS based on a four parameters values. The performance of the suggested method will be compared to other methods to prove its effectiveness in solving the tasks scheduling problem in the cloud environment.

1.11 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 provides earlier studies that are related to our work. Chapter 3 discusses in detail the methodology and the implementation stages of the proposed method to solve the problem of tasks scheduling based on the tasks attributes. Chapter 4 discusses in detail the results of our experiments, and compare the result of the proposed method with other existing algorithms. Chapter 5 concludes the thesis and produces the future work.

Chapter 2
Literature Review

2.1 Overview

Task scheduling problems are considered as one of the main challenges in the cloud environment. Many researches tried to solve the task-scheduling problem for the users' tasks. This chapter details some of the related work and a literature review to the problem being studies in this thesis.

2.2 Traditional Scheduling Algorithms in Cloud

There are many algorithms used to solve the task-scheduling problem like FCFS(Anuradha & Sumathi, 2014) . FCFS is a simple algorithm where task that arrives first will be scheduled first and resources are allocated to that task, as it need.

On the other hand, (Jia & Keung, 2013) proposed algorithm used SJF algorithm to be integrated with task grouping, priority-aware and SJF (shortest-job-first) to reduce the waiting time and make span, as well as to improve resource utilization.

Moreover, (Awan & Shah, 2015) proposed algorithm based on Round Robin method where the task is assigned to the VM. In this approach, a task is taken from the job pool. Then the select task is assigned to the available VM in round robin manner. This method has very less complexity with no overhead.

In general, Most of the traditional algorithms of scheduling in cloud computing such as FCFS, Round Robin, Min–Min and Max–Min scheduling algorithms do not take any consideration for the user needs and where the task is assigned to any available resource as soon as it arrives. Therefore, many researches are focused on optimize the task scheduling to reduce these problems.

2.3 Task scheduling in Private Cloud

Task scheduling in private cloud environment is quite complicated due to limited resources compared to the public cloud. In addition, the computational complexity and

the computing capacity of the processing elements are taken into account when scheduling the task in the private cloud environment. (Kumar & Verma, 2012) proposed an algorithm for private cloud environment that is based on the computational complexity and the computing capacity of the processing elements. The proposed algorithm using these parameters to build the algorithm that reduce the turnaround time and improve resource utilization.

In other researches, the researchers classified tasks based on the user priority based on their importance in a private cloud environment to give each user an appropriate priority to perform his tasks. **(The study in Singh et al, 2014**) proposed algorithm in a private cloud computing which has high throughput. The proposed algorithm classified tasks into two groups of users to reduce the average waiting time. The proposed algorithm also using bounded waiting for each group to overcome the problem of starvation.

2.3.1 Priority based Job Scheduling algorithm in Cloud

Priority refers to the fact of being regarded or treated with a task as more important than others. (Ghanbari & Othman, 2012) proposed an algorithm based on priority called (PJSC) by using mathematical demography. The proposed algorithm depends on multi-criteria such decision-making and mathematical model that known as Analytical Hierarchy Process (AHP). The proposed algorithm consists of three level priorities that include; scheduling level (objective level), resources level (attribute level) and job level (alternative level). The priority of each task is calculated and compared with other tasks separately. The suggested algorithm that provides priority for tasks that serve the decision-making, scheduling the tasks with minimum makespan and high throughput. However, the proposed algorithm is Inconsistency so it does not give optimal finish time. Moreover, (Saxena, Chauhan, & Kait, 2016) proposed priority based task scheduling algorithm where Tasks are given priority based on the size and VM are given priority based on Million Instructions per Second (MIPS). Then FCFS scheduling algorithm is used to allocate the tasks to the VMs.

2.3.2 Consistency Based Task Scheduling in Cloud

Consistency refers to agreement or harmony of parts or features to one another or a whole in Task Scheduling process in Cloud. (Ergu, Kou, Peng, Shi, & Shi, 2013) proposed algorithm that is based on analytical hierarchy process (AHP) focuses to improve consistency of comparison matrixes. Therefore, after the priority of each task is calculated, an appropriate cloud storage and cloud resources are assigned to the reciprocal tasks according to tasks weights. Although, the algorithm provides better approach to handle inconsistency, the approach still suffers from Over sighted finish time and complexity of algorithm issue.

2.3.3 Reduced Makespan Based Task Scheduling

Makespan is the whole execution time of the application tasks. (ROUHI & NEJAD, 2015) proposed algorithm called CSO-GA to reduce the Makespan for all the running tasks. The research presents a new meta heuristic scheduling technique using a combination of Cat Swarm Optimization (CSO) and Genetic Algorithm (GA). The proposed strategy reduce the Makespan compared to other techniques. Furthermore,(Arabnejad & Barbosa, 2014) presented a Heterogeneous Budget Constrained Scheduling (HBCS) algorithm. The algorithm aims to reduce the execution time and cost. The HBCS algorithm reduces the makespan by 30 % and the cost within the user's specified budget. Furthermore, the HBCS algorithm reduce the time complexity compared to other budget-constrained algorithms. In addition, (Chitra, Madhusudhanan, Sakthidharan, & Saravanan, 2014) used the JPSO algorithm to reduce the makespan. The JPSO algorithm overcome the problem of stocking in the local minima solution. The modified PSO algorithm make a jump in the $gbest$ value to avoid the poor convergence of the $gbest$ values. The results show that the proposed algorithm is more effective than the GA algorithm by 3.8% with a small number of tasks. However, the GA algorithm shows better result with a large number of tasks.

2.3.4 Cost Based Task Scheduling

Cost is the total cost of the application tasks executing over the VMs. (Selvarani & Sadhasivam, 2010) improves the traditional cost-based scheduling algorithm for making

appropriate mapping of tasks to resources. This algorithm groups the tasks according to the processing capabilities of available resources. The suggested algorithm enhanced the communication between tasks within each group and helped to provide the required resources for all tasks within each group at the same time. (Saxena et al., 2016) classifies the tasks into three categories according to the deadline and the cost constraints for each task and assign them to three levels of queues (High, Medium and Low). The approach is based on greedy resource (VM) allocation for selecting the resources in which the priority is given for the VM with minimum turnaround time for each individual task. Moreover, (Wu, Liu, Ni, & Gu, 2010) suggest Revised Discrete Particle Swarm Optimization (RDPSO) algorithm. The proposed algorithm is used to schedule the tasks over the different available resources. The Experiment is take place with a set of tasks with various data communication and computation costs based on the price model. The result showed that the proposed RDPSO algorithm could save cost and provide better makespan compared with the standard PSO and BRS (Best Resource Selection) algorithms. The proposed algorithm is not efficient with large search space.

2.4 Previous work

Researchers to schedule tasks on the resources have done several works. Each one had its own constraints that used to efficiently tasks scheduling.

(Kumar & Verma, 2012) proposed an algorithm, which assigns priority to different tasks based upon three parameters namely; tasks deadline, task age and the task length. After that, tasks are arranged in a sorted order by considering the calculated priority. Thus, the task with higher priority scheduled first

(Kumaresh, Prasidh, Arjunan, Subbhaash, & Sandhya, 2012) suggest a scheduling method based on organizes the subtasks based on the priority of the users to prevent the low priority jobs from starvation. Three levels of queues are used for each subtasks of users' priority (High, Medium and Low). The subtasks are assigned to the VMs in a round-robin manner through selecting three subtasks at the same time from the three levels of queues. The testing results indicate that the suggested algorithm enhance the

utilization rate of the VMs and provide high performance compared to the other scheduling algorithms.

(LIU & YANG, 2013) suggested an algorithm based on multi-QoS constraints and genetic algorithm to schedule tasks. The QoS is based on the execution time, cost or system load of the users' tasks. Therefore, The QoS of the users are differ from each other. the researcher gives users the ability to choose different scheduling goals according to their own needs. The suggested scheduling algorithm focus on the Scheduling process according to the users own needs. The results demonstrates that the proposed algorithm satisfies the QoS constrains, ensures the system load and enhances the performance of the task scheduling.

(Naseem, Al-Rahmawy, & Rashad, 2015) proposed an algorithm known as Performance and Cost Algorithm (PCA) depend on assigning tasks' priorities according to users' tasks profits, Three levels of queues are used to classify the users' tasks priority (High, Medium and Low). In addition, the aging technique with threshold value are used to avoid the infinite waiting of tasks in the lower queues. The aging technique moves tasks in the lower queues to the end of next high queue. The experiment showed that the suggested algorithm optimize the resource utilization through reducing the makespan which lead to high performance with low cost for the cloud users.

(Mohammed, 2016) proposed Best Level Job First algorithm based on four criteria; User level, Time, Cost and load on the system. The suggested algorithm adapt to the users level of the task and change its behavior in queue according to User level. The experiment results show that the proposed algorithm reduces the waiting time for the high-level users compared to the Short Job First (SJF) algorithm. In addition, the suggested algorithm fast the turnaround time for the high-level users compared to the Round Robin (RR) algorithm.

(Miao, 2016) introduced an innovative task scheduling and resource allocation strategy to improve the quality of service using thresholds with attributes and amount (TAM) in cloud computing. The attribute-oriented thresholds used to decide on the acceptance of tasks, and the provisioning of accepted tasks on appropriate (VMs,). The experimental results show that the suggested method improve attribute matching between tasks and VMs, with reduced the average execution time by 30 to 50% compared to the non-filtering policy.

(Saini & Kaur, 2017) proposed an algorithm in which the priority is given to different tasks according to specific attribute; User Level, Task urgency, Task Load and Time queuing up. Then, tasks are arranged in a sorted order by considering the calculated priority. Therefore, the task with higher priority scheduled first.

(Fadhil, 2017) proposed an algorithm called Best-Level-Job-First (BLJF) for private cloud computing. The user level is used as parameter with the other commonly used parameters in scheduling tasks. The tasks are classified according to the user level, and then priority is given to the tasks according to the tasks users. Also the algorithm can change its behavior by ignoring one or more parameters to satisfy the user need. The experiment results showed that the BLJF algorithm provides highest QoS to the users based on their levels.

(Ahmad, Ahmad, & Mirdha, 2017) introduced a new dynamic priority approach based job scheduling algorithm in cloud computing. The proposed model aims to reduce the waiting time, the turnaround time of tasks and to increase the throughput the system. The Aging technique is used to add an aging factor as a weight for each task. The simulation results can reduce the average waiting time, average turnaround time and total finish time of tasks. In addition, the Starvation problem is enhanced.

(Chugh, 2018) introduced several job scheduling algorithms and suggested a hybrid job scheduling algorithm to enhance the efficiency in cloud computing system.

The suggested hybrid algorithm consists of two phases. In the first phase the multi queue are managed and all the process over the different queues is being controlled. In the second phase, the different jobs are controlled over queues. The experiment results show that the proposed hybrid algorithm can reduce the waiting time for user jobs and enhance the throughput of the overall cloud environment.

The previous work shows that, there are many parameters that can influence the performance of cloud environment such as completion time, execution cost, VM load and user level. However, many researches have been ignored the influence of increasing the tasks that comes from low-level users. In this research, a new algorithm is proposed that is focused on scheduling users' tasks based on user priority level.

A Multi queue technique is used to classify the users' tasks priority where each user level has its own queue. In addition the tasks inside each queue schedule based on the priority of tasks. In addition, the priority scheduling algorithm to improve the Quality of service (QoS) for all users in the organization. HPJF also is compared with FCFS, SJF, RR and with the private scheduling algorithm (BLJF) to evaluate the performance of the overall cloud environment.

2.5 Chapter summary

In this chapter, the researcher presented a review of previous studies on tasks scheduling algorithms. This review indicates that there are different parameters can affect the cloud environment such as Execution Time, Cost, Load and user level. These parameters are used to improve QOS for the users and keep the system performance at an acceptable level. However, the effect of increasing the tasks that comes from low-level users is currently ignored as a problem in the researches. Therefore, a new scenario focuses on developing an algorithm to schedule users tasks that are based on user priority and to avoid the starvation of the low-level users' tasks.

Chapter Three
Methodology

3.1 Overview

This study was proposed for scheduling the tasks in private cloud to boost the productivity of the organizations that are based on user priority. The suggested scheduling algorithm classifies users requests into number of priority queues to give each user suitable priority according to his position in the organization. The tasks in the private cloud are exulted above other in some criteria such as execution time, execution cost and the load of the resource. The weighted summation of these criteria informs the weights for the tasks. The suggested scheduling algorithm also can schedules the tasks on a resource according to the tasks weights. This chapter illustrates the method followed in this research to develop HPJF algorithm. This chapter also gives details on the various steps of scheduling process based on the priority of the users, the suggested scheduling algorithm use the multi queue technique to overcome the starvation problem. HPJF algorithm is implemented using the Cloudsim simulator. The Cloudsim simulator is a well-known software tool for research in the area of resource allocation, provisioning, and task scheduling for cloud computing (Arabnejad & Barbosa, 2014). The Cloudsim simulator tool has been written in Java programming language. The Cloudsim simulator provides suitable environment for designing heterogeneous resources, apply different scheduling algorithms and measure their performance using the Eclipse IDE, which is integrated with the Cloudsim. The proposed algorithm is implemented using the following steps that are presented in this chapter. The results of this research are given and discussed in the next chapter.

3.2 The proposed algorithm (HPJF)

In the current study, the researcher suggests a scheduling algorithm based on user priority that is used to improve the performance of executing tasks in cloud environment. The proposed algorithm predicts the weight priority value for each incoming task to the VM. After that the task initially assigns to the VM with minimum weight priority. Then, the tasks are classified over multi-queue based on user priority and rearrange them based on the weight priority values. Finally, the tasks are distributed over the VM to be executed based on the user priority. The

overall design of the proposed algorithm is illustrated in Figure 3.1. The suggested scheduling algorithm functionality is described as follows:

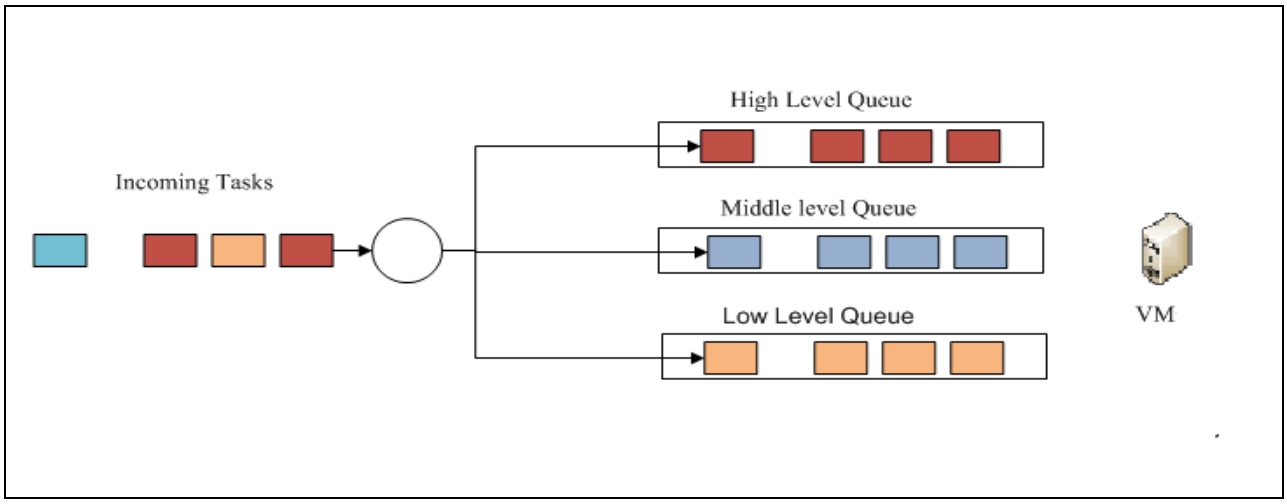


Figure 3.1: The overall design of the HPJF algorithm

3.2.1 Assign tasks into VM

In the first stage, the different types of users from different priority level users send their tasks to be processed in cloud environment. Each incoming task to the cloud has a specific attribute value that determines the priority of user, IO resource requirement, and CPU requirement and RAM requirement that are used to calculate the time, cost and load required for execute the task. In this research, CLOUDSIM simulator creates the tasks with different attribute values according to the equations calculations. After that, HPJF algorithm predicts the weight priority value for each incoming task to the VM. The weight priority value is calculated based on four attributes represented by user priority level, execution time, execution cost and VM load.

The Tasks priority weight is calculated based on considered attributes that are calculated previously (execution time, execution cost and VM load) as follow:

$$Priority\ weight = \omega_1\ Time + \omega_2\ Cost + \omega_3\ \text{Load} + \omega_4\ \text{User Level} \quad (1)$$

The parameters are arranged within each queue according to the values of the coefficients ratio that will be discussed in chapter four.

The execution time is calculated using equations (2) (Meng, Pappas, Li, 2010).

$$\text{Execution Time} = \text{cloudlet.getCloudletLength}() / \text{vm.getMips}() \quad (2)$$

The getCloudletLength is a function that returns the length of the task, and the getMips is a function that returns the size of the data file.

On the other hand, execution cost is calculated according to both resource price and the cost required for using the resource including CPU ($cost_{cpu}$), disk ($cost_{stor}$), memory ($cost_{ram}$) and bandwidth ($cost_{BW}$). The cost is expressed by the following equation (LIU, 2013):

$$cost = cost_{cpu} + cost_{ram} + cost_{BW} \quad (3)$$

After that the load is estimated using three parameters, which are the CPU usage (Load_cpu), the memory usage (Load_mem) and the use of the bandwidth rates (Load_br) (Heinze et al., 2013). Therefore, load is estimated to guarantee the load balancing among CPU, Memory and the bandwidth (Guang, Chen-Yang, Daoguoli, 2013).

The load is expressed by the following equation (LIU, 2013) :

$$Load = 1 - \prod_{k=1}^{3}(1 - Load_k)^{\omega Lk} \quad (4)$$

User Level is also added to the tasks weight in order to classify them at each queue.
The priority calculation process is a multi-objective function with a specific importance ratio for each objective represented by the coefficients ratio (ω) in the given function, to satisfy the applications requirement. For example, some users request a fast execution task, their tasks is chosen to be served first; by increase the ratio of time, while other users request tasks that don't required much money, their tasks are given priority to be served by increase the ratio of cost, and so on (Ji, Bao, & Zhu, 2017).

The function CalWeight is used to calculate tasks priority weight at each user based on execution time, execution cost and VM load attributes as follows:

Function: CalWeight ()
Input: Queue tasks **Output**: Tasks priority **For** Task_id =0 to last_task at the queue Priority weight = ω_1 Time + ω_2 Cost + ω_3 Load +ω_4 user priority return Priority weight **End For**

HPJF algorithm calculates the expected the weight priority value for each incoming task to be executed on VM. Then, the incoming task initially sorted according minimum weight priority value. As well as, the VM is allocated for the tasks that come from each user queue based their priority queue.

3.2.2 Tasks classification into queues

In this stage, HPJF checks the user priority attribute value for each incoming task to classify the tasks over the user queues. For the user priority parameter, the values are assumed between 1 and 10 to specify the user priority according to the user needs in the organization; in which each user level has its own queue.

There are two function that are used to distribute tasks to the priority queues; The function GetUserLevel is used to test the priority of user parameter values of the incoming tasks and The function ClassificationTasks is used to distribute the tasks to the appropriate queues as follows:

Function: GetUserLevel ()
Input: The incoming tasks **Output**: Type of user `return UserLevel;`

Function: ClassificationTasks()

Input: The incoming tasks

Output: *TaskGroup(1,2,3,4,5,6,7,8,9,10).*

```
for (intger i from 0 to tasks numbers)
      Task ta= (Task).getTasks(i);
      id=ta.getUser();
            if (id == 1) TaskGroup1.add(ta);// Manager user
            else if (id == 2) TaskGroup2.add(ta);
            else if (id == 3) TaskGroup3.add(ta);
            else if (id == 4) TaskGroup4.add(ta);
            else if (id == 5) TaskGroup5.add(ta);// Middle user
            else if (id == 6) TaskGroup6.add(ta);
            else if (id == 7) TaskGroup7.add(ta);
            else if (id == 8) TaskGroup8.add(ta);
            else if (id == 9) TaskGroup9.add(ta);
            else
              TaskGroup10.add(ta);// End users
```

End For

Therefore, the tasks that are sent from the managers, placed in the high-level queue. Whereas, the tasks from users who have middle priority are located in the middle priority queues. As well as, the tasks from users who have low priority are located in the low priority queues.

3.2.3 Sort tasks within each queue

After distribute the tasks into different priority queues (High, middle and low) based on user priority, the tasks are sorted in ascending order within each queue based on the weight values of each task. Therefore, the task which has the lowest priority weight value is the first task will be sent to the VM to be executed.

3.2.4 Send tasks to VM

The suggested algorithm uses the multi queue technique to reduce starvation in a scheduling system. The multi queue technique works by give each queue a chance to use the resource in which large number of tasks is taken from the high priority queue and the small number of tasks is taken from the lower priority queue. The scheduler takes 10 tasks from manager queue then 9 tasks from Q9 and so on by using the function as shown below

Function: CloudletSchedulling ()

Input: *TaskGroup1*, *TaskGroup2*, *TaskGroup3*, *TaskGroup4*, *TaskGroup5*, *TaskGroup6*, *TaskGroup7*, *TaskGroup8*,*TaskGroup9*, *TaskGroup10*

Output: Rotate scheduler to the priority queue in round robin fashion.

while size(*TaskGroup1*) not 0 or size (*TaskGroup2*) not 0 or size(*TaskGroup3*)!= 0
or size (*TaskGroup4*) not 0 or size (*TaskGroup5*) not 0 or size(*TaskGroup6*)!= 0
or size (*TaskGroup7*) not 0 or size (*TaskGroup8*) not 0 or size(*TaskGroup9*)!= 0
or size (*TaskGroup10*) not 0

if(j=1 And size (*TaskGroup1*) not 0){ schedule (*TaskGroup1,10*) ;count++ **End If**
if(j=2 And size (*TaskGroup2*) not 0){ schedule (*TaskGroup2, 9*) ;count++ **End If**
if(j=3 And size (*TaskGroup3*) not 0){ schedule (*TaskGroup3, 8*) ;count++ **End If**
if(j=4 And size (*TaskGroup4*) not 0){ schedule (*TaskGroup4, 7*) ;count++ **End If**
if(j=5 And size (*TaskGroup5*) not 0){ schedule (*TaskGroup5,6*) ;count++ **End If**
if(j=6 And size (*TaskGroup6*) not 0){ schedule (*TaskGroup6, 5*) ;count++ **End If**
if(j=7 And size (*TaskGroup7*) not 0){ schedule (*TaskGroup7,4*) ;count++ **End If**
if(j=8 And size (*TaskGroup8*) not 0){ schedule (*TaskGroup8,3*) ;count++ **End If**
if(j=9 And size (*TaskGroup9*) not 0){ schedule (*TaskGroup9,2*) ;count++ **End If**
if(j=10 And size (*TaskGroup10*)not 0){ schedule (*TaskGroup1,1)* ; count= 1) **End If**

End while

As well as, the scheduler take number of tasks from each queue in each rotation according to user priority for example, the scheduler takes 10 tasks from the manager queue and one task from the End user queue. Moreever, each queue has appropriate threshold value. If the queue size less than the threshold value of that queue, Note that the threshold values for each queue have been tested in chapter four to show which is the best threshold value for each queue. Therefore, the whole tasks in the queue will be taken, using the function Schedule () as shown below:

Function: schedule (*TaskGroup*, *tasksNo*)

Input: *TaskGroup*, *tasksNo*
Output: assign tasks to VM
J= *TaskGroup.Size*
If (j < *tasksNo)*
For Task_id =0 to *tasksNo*
Add Task_id to *FinalGroup*
Add Task_id from *TempGroup*
End For
For Task_id =0 to *tasksNo*
Remove Task_id from *TaskGroup*
Remove Task_id from *TempGroup*
End For
End If
Else
For Task_id =0 to *TaskGroup.Size*
Add Task_id to *FinalGroup*
Add Task_id to *TempGroup*
End For
For Task_id = 0 to *TaskGroup.Size*
Remove Task_id from *TaskGroup*
Remove Task_id from *TempGroup*
End For
End Else

3.3 An example for the proposed algorithm

Let us as considered there are 20 incoming tasks that are generated by different priority level of users as in table 3.2. The values 1, refers to the highest priority users. Whereas, 5 refers to middle priority users. While, 10 values, refers to Lowest priority users. The tasks in this example are generated from the user level 8, 9 and 10.

The suggested algorithm is implemented in the following steps:

Step 1: Assign tasks into VM

In the first step, HPJF algorithm calculates the weight value for each incoming task over all the VM. The weight priority value is calculated based on three attributes represented by execution time, execution cost and VM load using equation (1), (4), (5) and (6) as well as the user priority attributes. The results of this step can be shown in Table 3.1.

Table 3.1: Incoming tasks by different priority level of users

Task	User Priority	Time	Cost	Load	Weight
9	9	2.191	0.14	0.0078	4.2859
1	8	1.077	0.13	0.023	3.5508
18	9	1.992	0.17	0.0275	4.2338
15	10	2.116	0.16	0.0037	4.6674
2	10	2.203	0.1	0.0074	4.6817
11	9	2.976	0.2	0.0116	4.5337
14	8	2.554	0.21	0.0153	4.0097
17	8	1.034	0.19	0.0149	3.5498
12	9	1.166	0.14	0.0073	3.9783
6	8	1.55	0.09	0.0236	3.6858
0	10	1.531	0.13	0.0039	4.4861
13	9	1.125	0.17	0.0186	3.974
7	10	2.706	0.21	0	4.8531
4	9	1.315	0.17	0.0186	4.0309
16	10	2.571	0.21	0.0272	4.8156
19	8	2.071	0.08	0.0075	3.8389
5	10	1.646	0.16	0.0037	4.5266
10	8	2.678	0.14	0.0268	4.0332
8	8	2.781	0.15	0.0074	4.0646
3	9	2.046	0.14	0.0109	4.2431

Step 2: Tasks classification into queues

The suggested algorithm uses the function getUser to examine the user priority of the incoming tasks to send the tasks to the appropriate queues so all the tasks with user priority equal to 8 will be located in the highest queue as shown in Table 3.2.

Table 3.2 Tasks classification into highest-level queue

Task id	User Priority	Time	Cost	Load	Weight
1	8	1.077	0.13	0.023	3.5508
14	8	2.554	0.21	0.0153	4.0097
17	8	1.034	0.19	0.0149	3.5498
6	8	1.55	0.09	0.0236	3.6858
19	8	2.071	0.08	0.0075	3.8389
10	8	2.678	0.14	0.0268	4.0332
8	8	2.781	0.15	0.0074	4.0646

On the other hand, all the tasks with user priority equal to 9 will be located in the second queue as shown in Table 3.3.

Table 3.3 Tasks classification into middle level queue

Task id	User Priority	Time	Cost	Load	Weight
9	9	2.191	0.14	0.0078	4.2859
18	9	1.992	0.17	0.0275	4.2338
11	9	2.976	0.2	0.0116	4.5337
12	9	1.166	0.14	0.0073	3.9783
13	9	1.125	0.17	0.0186	3.974
4	9	1.315	0.17	0.0186	4.0309
3	9	2.046	0.14	0.0109	4.2431

Furthermore, all the tasks with user priority equal to 10 will stored in the third level queue as shown in Table 3.4.

Table 3.4 Tasks classification into low-level queue

Task Id	User Priority	Time	Cost	Load	Weight
15	10	2.116	0.16	0.0037	4.6674
2	10	2.203	0.1	0.0074	4.6817
0	10	1.531	0.13	0.0039	4.4861
7	10	2.706	0.21	0	4.8531

16	10	2.571	0.21	0.0272	4.8156
5	10	1.646	0.16	0.0037	4.5266

Step 3: Sort tasks within each queue

After distribute the tasks over the user priority queues, the tasks are rearranged within each queue based on the priority weight of each task to be ready to execute on the VM. The results for the user priority 8, 9 and 10 show as in Table 3.5, Table 3.6 and Table 3.7 respectively.

Table 3.5: Rearrange tasks in high-level queue based on priority weight value

Task id	User Priority	Time	Cost	Load	Weight
17	8	1.034	0.19	0.0149	3.5498
1	8	1.077	0.13	0.023	3.5508
6	8	1.55	0.09	0.0236	3.6858
19	8	2.071	0.08	0.0075	3.8389
14	8	2.554	0.21	0.0153	4.0097
10	8	2.678	0.14	0.0268	4.0332
8	8	2.781	0.15	0.0074	4.0646

Table 3.6: Rearrange tasks in middle-level queue based on priority weight value

Task id	User Priority	Time	Cost	Load	Weight
13	9	1.125	0.17	0.0186	3.974
12	9	1.166	0.14	0.0073	3.9783
4	9	1.315	0.17	0.0186	4.0309
18	9	1.992	0.17	0.0275	4.2338
3	9	2.046	0.14	0.0109	4.2431
9	9	2.191	0.14	0.0078	4.2859
11	9	2.976	0.2	0.0116	4.5337

Table 3.7: Rearrange tasks in low-level queue based on priority weight value

Task Id	User Priority	Time	Cost	Load	Weight
0	10	1.531	0.13	0.0039	4.4861
5	10	1.646	0.16	0.0037	4.5266
15	10	2.116	0.16	0.0037	4.6674
2	10	2.203	0.1	0.0074	4.6817
16	10	2.571	0.21	0.0272	4.8156
7	10	2.706	0.21	0	4.8531

Step 4: Send tasks to VM

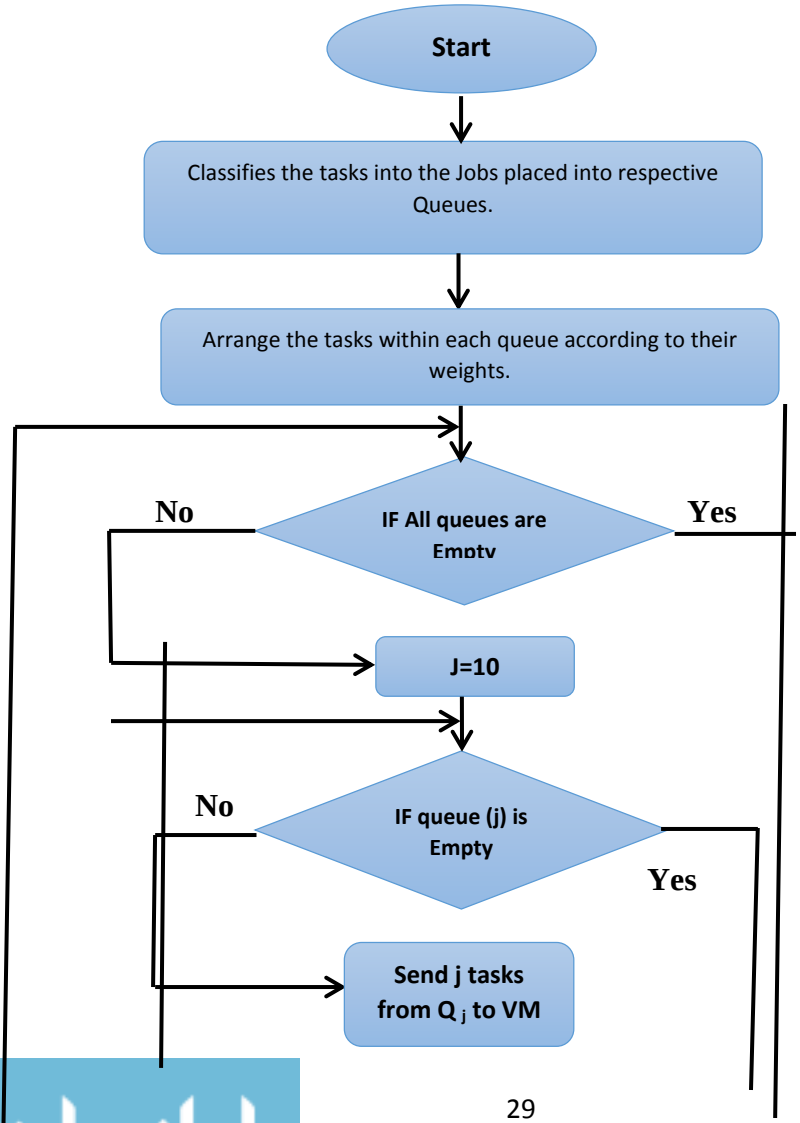
As discussed previously in step 4.3.1, the scheduler begins its loop from the highest priority queue and continues in rotation among the available queues in a round robin fashion. The scheduler takes a number of tasks from each queue according to the priority for the users. The tasks will be moved to the VM as shown in Table 3.8.

Table 3.8: Tasks classification after applying HPJF technique

Task	User Priority	Time	Cost	Load	Weight
17	8	1.034	0.19	0.0149	3.5498
1	8	1.077	0.13	0.023	3.5508
6	8	1.55	0.09	0.0236	3.6858
13	9	1.125	0.17	0.0186	3.974
12	9	1.166	0.14	0.0073	3.9783
0	10	1.531	0.13	0.0039	4.4861
19	8	2.071	0.08	0.0075	3.8389
14	8	2.554	0.21	0.0153	4.0097
10	8	2.678	0.14	0.0268	4.0332
4	9	1.315	0.17	0.0186	4.0309
18	9	1.992	0.17	0.0275	4.2338
5	10	1.646	0.16	0.0037	4.5266
8	8	2.781	0.15	0.0074	4.0646

3	9	2.046	0.14	0.0109	4.2431
9	9	2.191	0.14	0.0078	4.2859
15	10	2.116	0.16	0.0037	4.6674
11	9	2.976	0.2	0.0116	4.5337
2	10	2.203	0.1	0.0074	4.6817
16	10	2.571	0.21	0.0272	4.8156
7	10	2.706	0.21	0	4.8531

Finally, the algorithm uses a variable to determine the number of tasks that is taken from each queue in each rotation where the value of the variable is equivalent to the queue identification the implementation of the proposed Algorithm is represented by the following flowchart to explain the methodology for this algorithm as shown in figure 3.7.



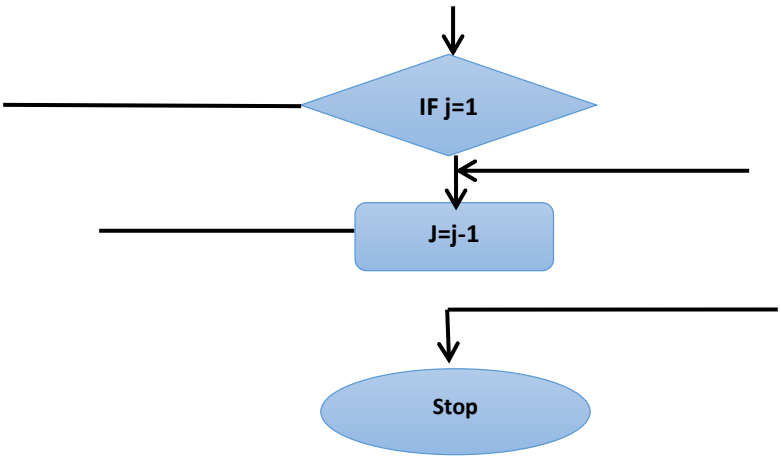


Figure 3.2: Flowchart for the HPJF Algorithm

Chapter Four
Simulation results

4.1 Overview

As presented previously, this thesis presents a new task scheduling strategy based on multi-level queue allowing high tasks to move up to higher queue propriety levels. It is expected that HPJF algorithm will reduce waiting and turnaround times for the tasks, improving cloud performance. The performance of HPJF algorithm is compared with the performance of the traditional and well-known scheduling algorithms FCFS, RR, SJF, and also with the private scheduling algorithm BLJF (Fadhil, 2017). FCFS and SJF have been chosen because they are standard algorithms, and also because FCFS is fair and it is widely used in other similar studies (Ababneh, 2006). RR also is an algorithm that give each task equal chance (time slice) to get a resource in which all tasks have the same priority (Sotomayor, Montero, Llorente, & Foster, 2009). BLJF is a recent scheduling algorithm for private clouds and it is also based on user priority. For this work, a cloud simulation tool is selected so as to mimic a real cloud environment and evaluate the performance of task scheduling and resource allocation with the proposed strategy.

4.2 Introduction to simulation used

In this thesis, the method of the study is simulation because the cloud environment is large and complex. Cloud users have different Quality of Service (QoS) requirements and the cloud itself needs to accommodate varying demands. It is difficult to perform tests using a real-world Cloud (Buyya,et al, 2010). This is because the system may not exist, take much time to build, and be costly to build. In the simulation, many experiments have been conducted to examine the effect of moving tasks between queues.

The CloudSim simulator has been selected to implement the proposed algorithm. It is a well-known software that has been written in JAVA; a powerful object-oriented programming language (Ahmad and Sabyasachi, 2014). CloudSim has been widely used for evaluating various algorithms in the area of resource allocation, provisioning, and task scheduling for cloud computing (Arabnejad & Barbosa, 2014).

The main components of CloudSim are the Datacenter, Host, Virtual machines (VMs), and Datacenter Broker. A datacenter consists of a set of hosts, where a host represents the physical computing node in a cloud. Each host consists of a set of virtual machines and is responsible for managing VMs during their life cycle. The datacenter broker is responsible for load balancing of VMs and managing the routing of user tasks among data centers based on different policies (Buyya, et al, 2010).

4. 3 Performance Evaluation Factors and Criteria for the HPJF Algorithm

To assess the performance of the proposed algorithm, some factors for describing the evaluation criteria are needed.

For the user priority factor in the proposed algorithm, the number of user priority levels are assumed to be in the range 1 to 10, where 1 represents the highest user priority level and 10 the lowest user priority level.

The values of weighting coefficients vary from 0 to 1 for the following factors that are used to calculate the task weights, TWs. The TWs are used to sort tasks in the queues.

TW= $User\ priority * \omega_1$+Execution time $* \omega_2$+Execution $cost * \omega_3$+ Load*ω_4

ω_1+$\omega_2 + \omega_3 + \omega_4$=1, where the values of these coefficients ratio is 0.4, 0.3, 0.2 and 0.1 respectively.

The coefficients ratio ω_1 is the highest value because this is the main goal of the algorithm HPJF. Therefore, The tasks that have the same weight are sorted according to their initial order.

The aim of the simulations is to evaluate the factors that affect pricing in a private cloud. The simulated private cloud environment consists of one datacenter and one broker. The datacenter uses x86 architecture and running in a Linux operating system and Xen virtual machine manage

The following table includes the specifications for both VM and Cloudlet in the datacenter:

Table 4.1: the specifications for both VM and Cloudlet.

VM		**Cloudlet**	
CPU	1000 MIPS	Length	[1000,3000] MIPS
RAM	1024 MB	File size	[300,1000] MB

Storage	1GB	Output size	[300,1000] MB
Bandwidth	1000 bps	Number of task	[100,1000]

The development environment for the simulation experiments has been described in the previous chapter. It is CloudSim 3.0.3, JDK 18 and Eclipse Kepler Server Release 1. The experiments were performed on a Windows 7 (64-bit), Intel core i3-2365M CPU, 1.40 GHz Processor, 4.0 G installed memory, and 500GB Hard disk drive.

The main performance parameters used are Average Waiting Time (AWT) and Average Turnaround Time (ATT) of tasks. The Turnaround time of a task is the time that the task spends in the system from arrival to departure, while the waiting time of a task is the time that the task spends in the queue before the task starts execution.

4.3 Experimental Results and Discussion

To perform simple but typical experiments, one virtual machine and the number of tasks varying from 100 to 1000. The tasks are randomly generated and submitted to the private cloud system. This is done on the assumption that there are 10 priority levels of users in the institution that uses the private cloud system. Among the tasks under the same user priority level, some tasks are exulted above other in some criteria. For example, for the tasks that request computation Service, they will be schedule first, for the tasks that request an inexpensive Service; they will be schedule second and so in order to satisfy the organization needs.

4.3.1 Applying Threshold to each user level queue

Priority scheduling can suffer from a major problem known as indefinite blocking, or starvation, where a low-priority task can wait a very long time because there are always some other tasks around that have higher priority.

Applying the multi-level queue technique may reduce the tasks waiting time in which each user level has its own priority queue. Moreover, each user level is allowed to use the resource according to the priority values for the user level queue. These values represent the maximum

number of tasks (threshold) that send to the resource in each rotation.

Therefore, applying the multi-level queue technique indicates that the average time a task spends in the run queue is reduced, which leads to reducing task starvation.

.4.3.2The Evaluation results for each user priority level:

Figures **4.1** and **4.2** represent the simulation results for both the average waiting time and average turnaround time when they are plotted against number of user levels for different task scheduling algorithms such as FCFS, SJF, RR, BLJF, and HPJF algorithm. The number of tasks in this experiment is 1000 tasks submitted to the private cloud system.

Based on the results, it can be inferred that the performance of HPJF algorithm is better than that of the other traditional scheduling algorithms on some user priority levels. The performance of HPJF algorithm is better than FCFS from point 1 to 7 and SJF from point 1 to 6. The performance of HPJF algorithm is also better than RR from point 1 to 9 and is better than BLJF from point 2 to 10. This is because in HPJF algorithm the tasks are sortied according to user priority while FCFS, RR and SJF do not consider user priority. Moreover, the improvement in AWT and ATT is due to allowing starved tasks with low levels of propriety to be executed with high priority tasks.

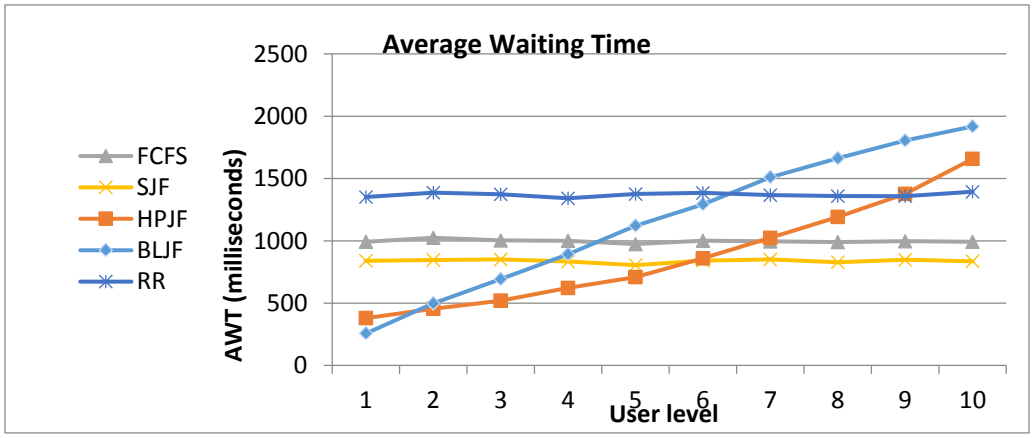


Figure 4.1 Average Waiting Time vs. user level

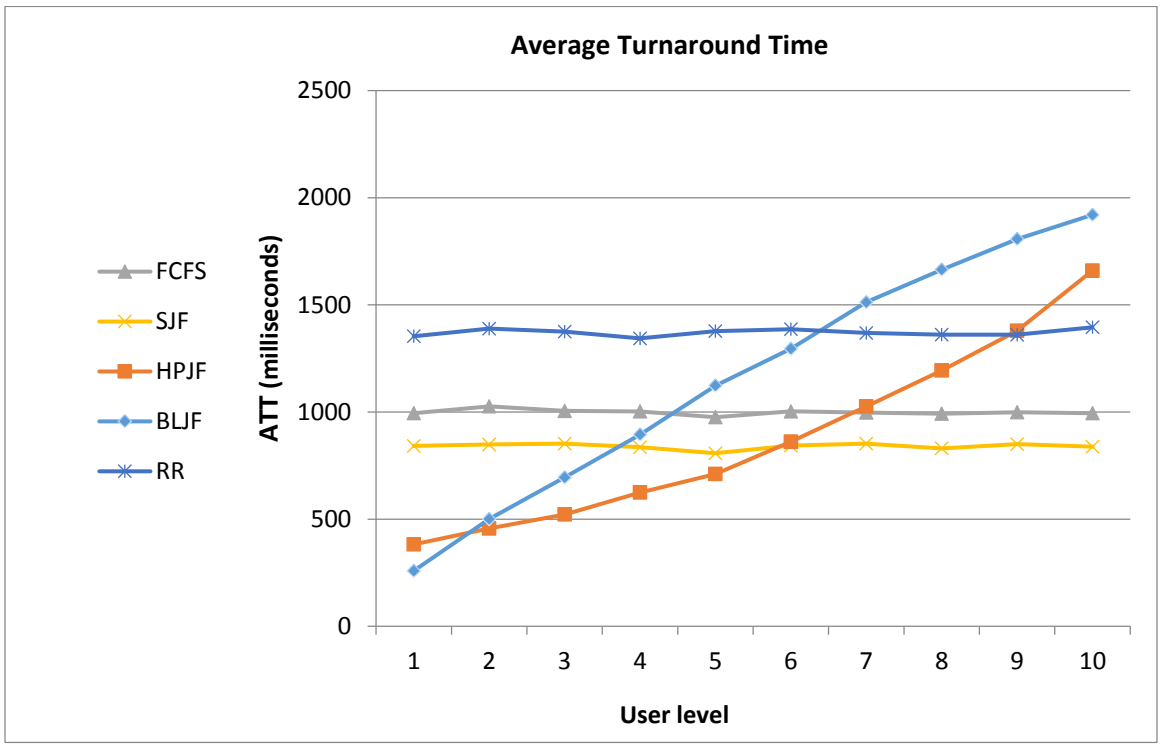


Figure 4.2 Average Turnaround Time vs. user level

4.3.3 The Evaluation results when the tasks number is increased:

In this section, we consider the same behavior of HPJF algorithm as discussed previously in chapter three and we will test a number of experiments when the numbers of tasks is changed to stand on the performance of the algorithm HPJF.

Figures 4.3 and 4.4 display the simulation results (AWT and ATT) that have been measured for the five scheduling algorithms BLJF, SJF, FCFS,RR and HPJF algorithm for different numbers of tasks using the same number of levels of users (10 levels of users).

From the results, it can be noticed that the performance of HPJF algorithm is better than that of the traditional scheduling algorithms (FCFS and RR) and the private scheduling algorithm (BLJF) when the number of tasks submitted to the cloud server increased gradually. This means that using multi queue and priority scheduling do not affect negatively on the algorithm and

gives positively slated results for the system. The results can be seen in the figures below.

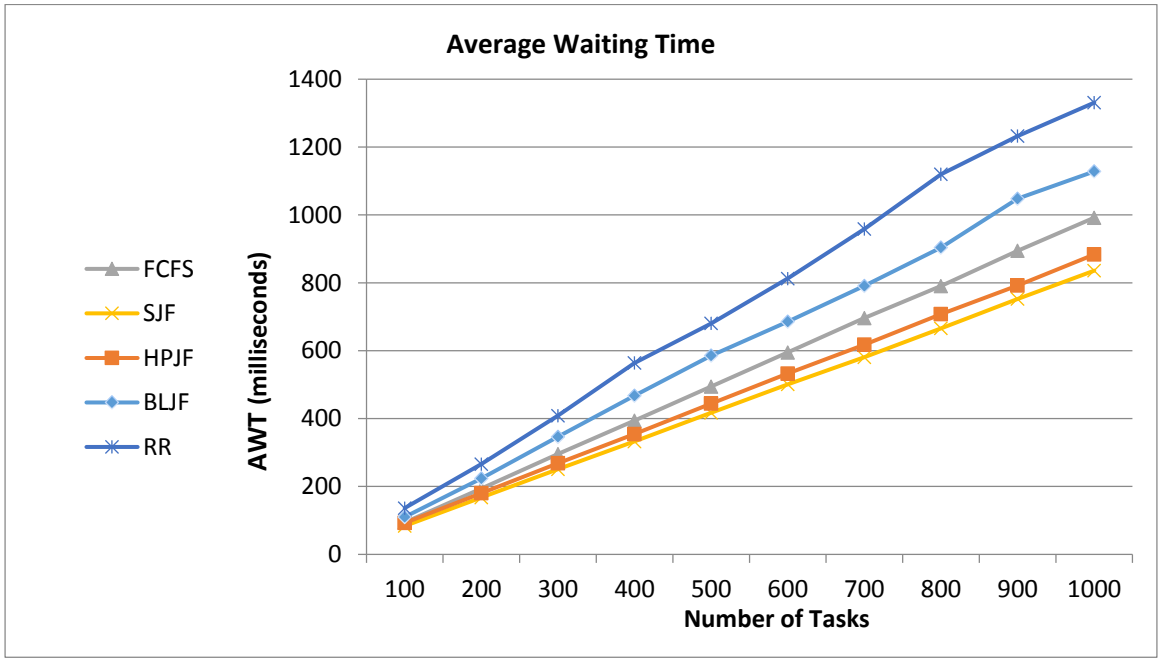


Figure 4.3 Average Waiting Time vs. Number of tasks

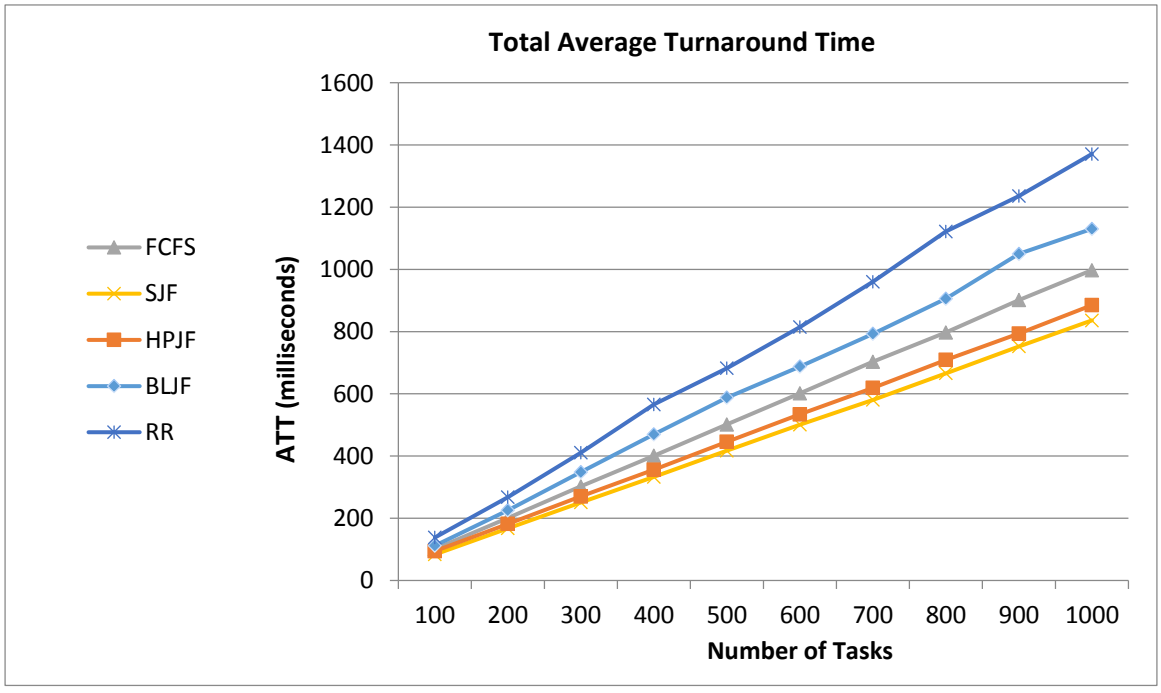


Figure 4.4 Average Turnaround Time vs. Number of tasks

.4.3.4 The Evaluation results when the number of user levels is increased:

We have seen in Figures 4.5 and 4.6 that the performance of the proposed Algorithm is better than that of BLJF in terms of both AWT and ATT, and it is better especially when the number of user levels is small. This is because the opportunity is given to the other factors to be sorted according through, where the tasks under the same user level is sorted according to their execution time at first, if the tasks have the same user level and execution time and they will be sorted according to their cost and so on. However, AWT and ATT increase gradually as the number of user levels becomes large; this is because the small number of tasks becomes under the same user level when the number of user levels increased which may not be sorted according to the other factors such as execution time, cost of execution and system load.

In the worst case, when each task comes from different user level (this is abnormal situation), then it will take the largest waiting time and turnaround time because tasks will be sorted only according to the user level, and tasks are not given the opportunity to be sorted according to the other factors.

From the results, it can be noticed that the number of user levels is important for the performance efficiency. However, in some cases, the user level factor gives inferior results on the performance when the number of user levels becomes very large for such organizations.

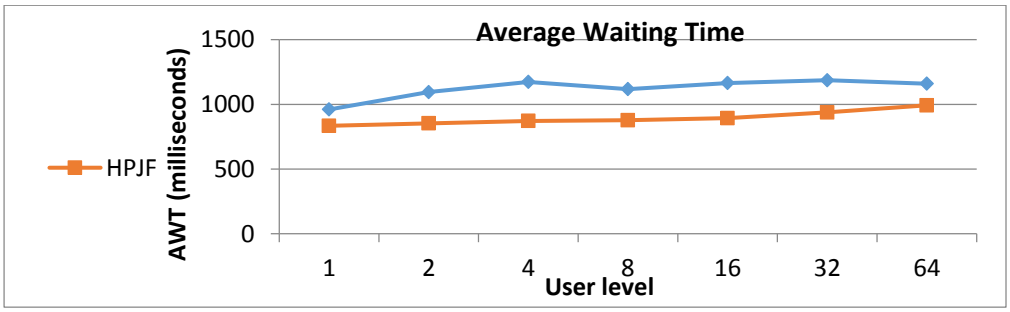


Figure 4.5 Average Waiting Time vs. Number of user levels

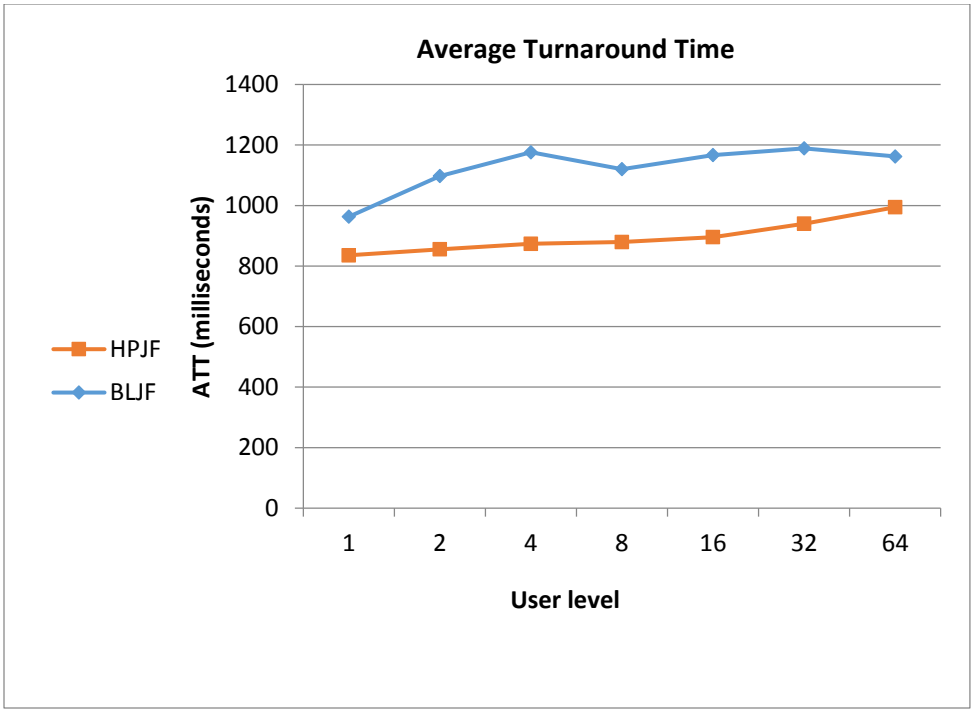


Figure 4.6 Average Turnaround Time vs. Number of user levels

4.3.5. The Evaluation results when the behavior of HPJF is changed:

In this section, we consider the behavior of HPJF algorithm as discussed previously in chapter three and we will test a number of cases to change the behavior to show which of that cases is better. The followed method in our work is that when the scheduler takes ten tasks from manager queue (Q10) and takes nine tasks from the lower queue (Q9) and so on until it reach to the lowest queue one (Q1); the scheduler will take one tasks from it.

We have seen in Figures 4.7 and 4.8 that the performance of HPJF Algorithm is better in terms of both AWT and ATT when the number of tasks number that is taken from each queue in each rotation is very small. This is because the lower queues don't wait for a long time for scheduling their tasks to avoid the starvation problem.

From the results, it can be noticed that the number of tasks number that is taken from each queue in each rotation is important for the performance efficiency. However, in the worst case,

the behavior gives inferior results on the performance when the number of tasks number that is taken from each queue is very large.

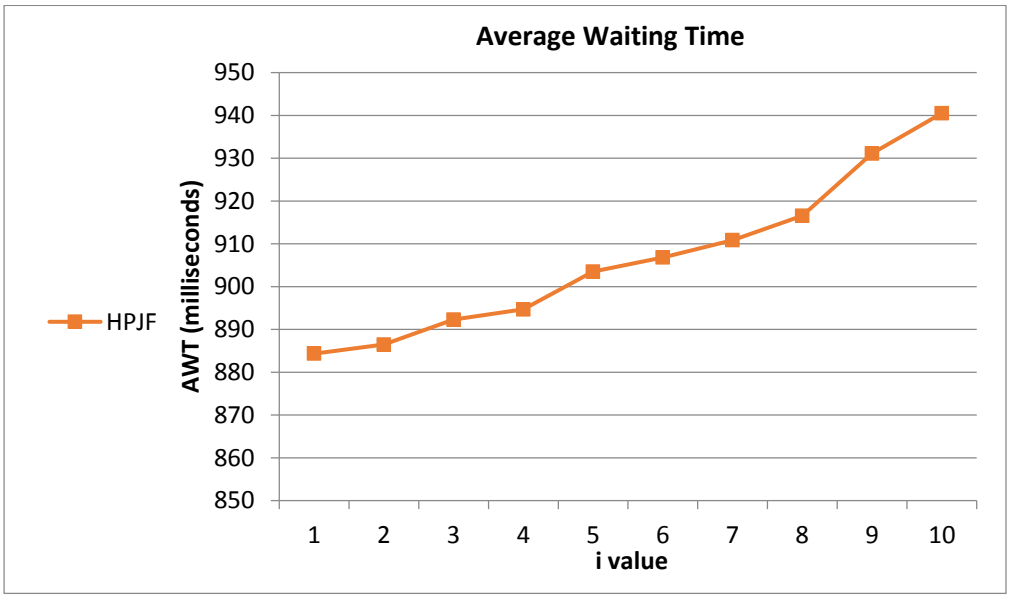


Figure 4.7 Average Waiting Time vs. i*Number of tasks taken from each queue

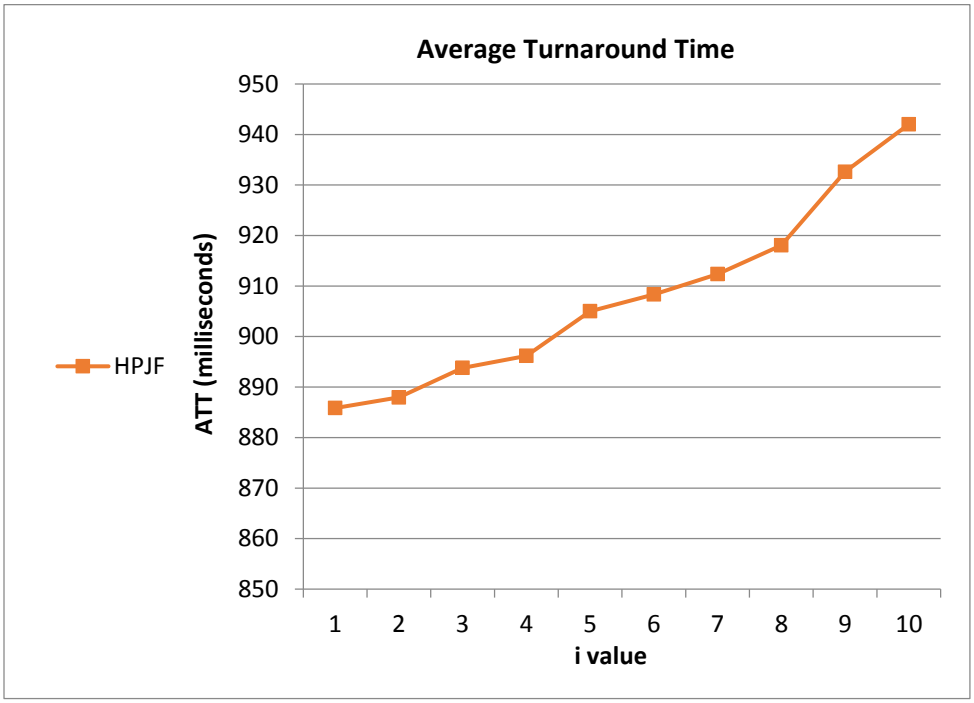


Figure 4.8 Average Turnaround Time vs. i*Number of tasks taken from each queue

4.3.6. The Evaluation results when the order of parameters is changed:

In this section, we consider set of experiments that effect on the behavior of HPJF algorithm. These experiments lead to replace the order of parameters to show which of that experiments is better. The followed order in our work is represented by experiment one (Exp1) in table below.

In Table 4.2, HPJF algorithm is tested by set experiments (Exps) to test the ability of the algorithm HPJF, each experiments uses different values of the coefficients ratio for the parameters used. This will change the values of the tasks' weights and certainly lead to change performance of HPJF algorithm. This is done on the assumption that the values of w1, w2, w3, and w4 are referred to the coefficients ratio for the Execution Time, Cost, Load, and user level respectively. We also assumed the parameter which has the highest the coefficients ratio is that the parameter selected to be first. 'User Level' parameter has always the highest coefficients ratio because this is the main goal of the algorithm HPJF.

We have seen in Figures 4.9 and 4.10 that the performance of HPJF Algorithm is better in terms of both AWT and ATT when the order of the parameters is taken from Exp1. This is because that which the same user level are sorted according to their execution time, cost and load respectively.

Table 4.2: set of the experiments uses different values of the coefficients ratio.

Performance measure	(W1)	(W2)	(W3)	(W4)
Exp1	0.3	0.2	0.1	0.4
Exp2	0.3	0.1	0.2	0.4
Exp3	0.2	0.3	0.1	0.4
Exp4	0.1	0.3	0.2	0.4
Exp5	0.2	0.1	0.3	0.4
Exp6	0.1	0.2	0.3	0.4

From the results, it can be noticed that the order of parameters within each queue is important for the performance efficiency. However, in the worst case, the behavior gives inferior results on

the performance when the parameters are arranged according to Exp5.

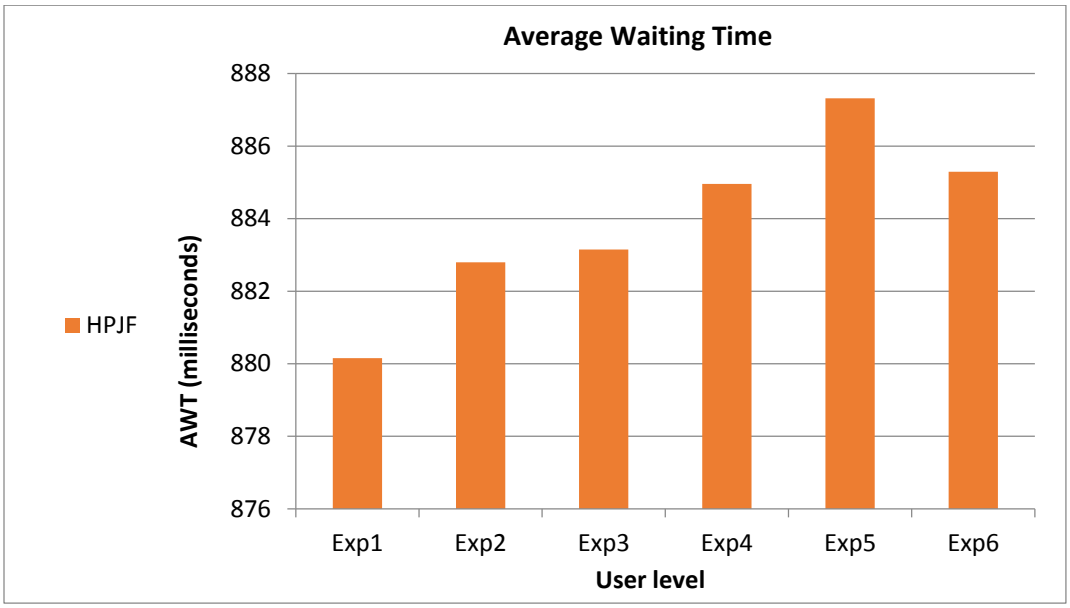


Figure 4.9 Average Waiting Time vs. the experiments in Table 4.2.

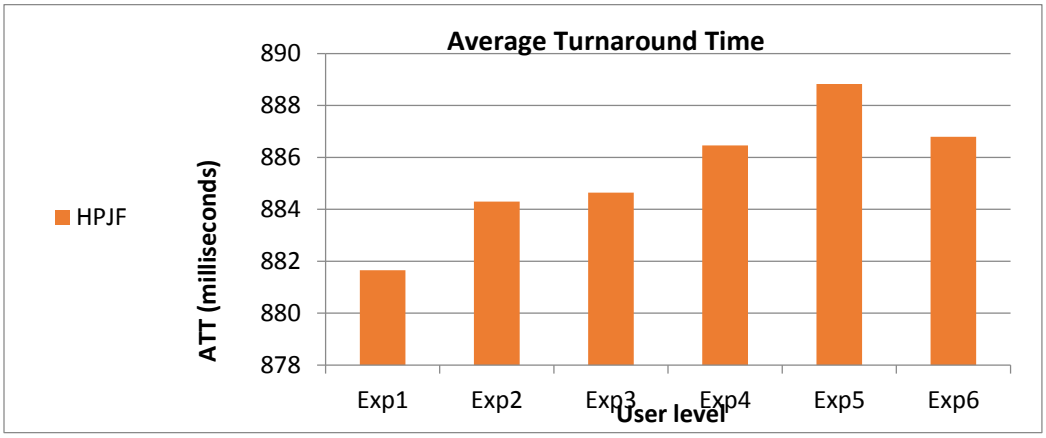


Figure 4.10 Average Turnaround Time vs. the experiments in Table 4.2

4.3.4 The evaluation results for each scheduling algorithm:

In this section, we consider the same scenario as discussed previously in Figures 4.1 and 4.2 using the same number of levels of users and the same number of tasks. The simulation results for the AWT and ATT for all levels can be presented in the form of groups to show which task scheduling algorithm is better and also to show which of the five scheduling algorithms has inferior results.

Table 4.3: Average Waiting Time and Average Turnaround time for each scheduling Algorithm.

Performance measure	The Proposed	BLJF	SJF	FCFS	RR
AWT	883.663301	1128.234	833.9167	991.8011	1331.086
ATT	885.16416	1130.22942	835.919	997.3852	1370.975436

In Table 4.3, HPJF algorithm succeeds in reducing the average waiting time for all user priority levels in comparing with BLJF and RR. The waiting time for HPJF algorithm is 883.663301, which is approximately 21.6%, 33.60%, and 10.9% better than BLJF, RR, and FCFS respectively. The results can be analyzed using the bar chart shown in Figure 4.11.

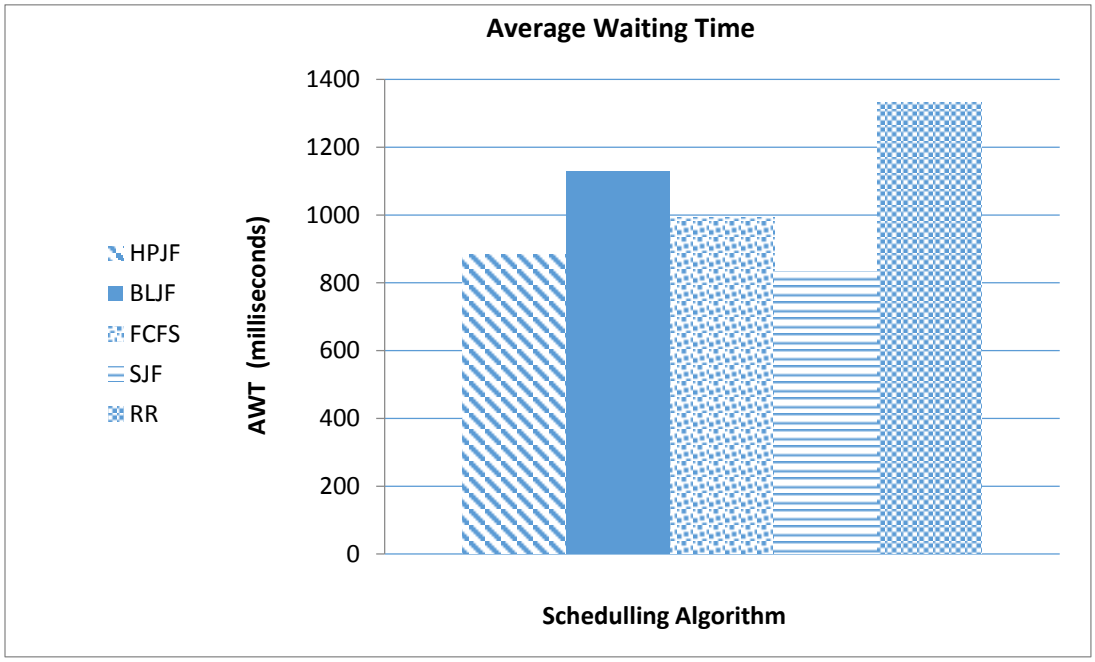


Figure 4.11 Average Waiting Time vs. scheduling Algorithms

In the table shown previously, HPJF algorithm also succeeds in reducing the average Turnaround time at over user priority levels. The average Turnaround time for HPJF algorithm is 885.16416, which is approximately 21.68%, 35.40%, and 11.25% better than BLJF, RR, and FCFS respectively. The results can be analyzed using the bar chart shown in Figure 4.12.

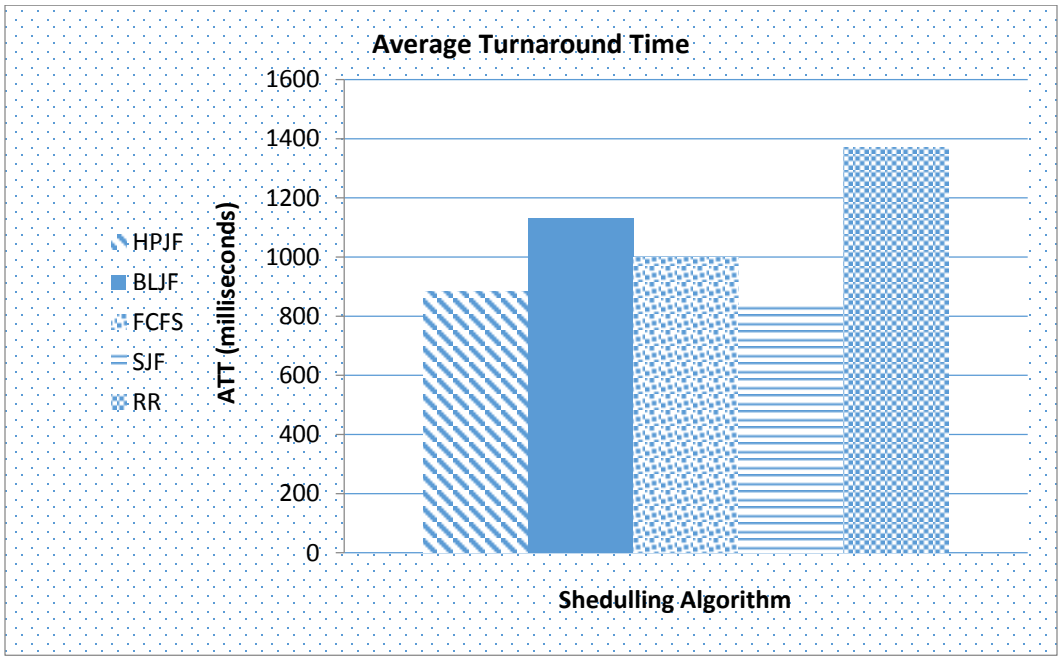


Figure 4.12 Average Turnaround Time vs. scheduling Algorithms

4.4 Summary

The results presented in this chapter show that using multiple priority queues and applying the priority technique could boost the productivity of organizations. The performance of the proposed policy has been compared against that of the existing FCFS, SJF, RR and BLJF policies. The simulation results show that HPJF algorithm can significantly improve performance of the system in general. The performance of HPJF algorithm is better at both lower and middle levels in particular. However, the performance of the system is not better from others on all levels but it is not return negative result. In the future, it will be better to find a technique to improve the performance at all user levels.

Therefore, Reducing the waiting time indicates that the average time a task spends in the waiting queue is reduced which leads to reducing task starvation through allowing low priority tasks to be executed with high priority. This helped to produce tasks scheduling algorithm in private cloud computing for the organizations that are based in user priority.

Chapter Five
Conclusion and Future Work

5.1 Conclusion

Task scheduling is one of the major researches which play a major role of improving the performance of the cloud system by arrangement the tasks in way that satisfy the user requirements. An efficient task scheduling algorithm is that which reduce waiting time Turnaround time for the tasks.

In this thesis, the motivation of task scheduling is preserved many lower priority tasks are starved when the execution priority is given to the higher priority tasks, a task may waits for a very long time to be executed. Starvation is frequently brought on by lapses in a scheduling calculation.

The experiment results indicates that proposed approach has produced better results in term of overall average waiting and overall average Turnaround time over the BLJF and the other traditional scheduling Algorithms. Although HPJF algorithm shows better result for high and middle priority user levels, still there is a need to enhance the results. In future, the results can be improved at all levels.

5.2 Directions for the Future Works:

There are several interesting issues and open problems that worth further investigation. Some of them are briefly described below.

1- The proposed strategy has been shown to perform well in independent tasks. It would be interesting to adapt the dependent tasks or event that both dependent and independent tasks which will certainly affect the performance on Cloud system.

2- In this research, the tasks based user priority can be tested after clustering the resources based on user priority which will certainly affect the performance on Cloud System.

3- The proposed strategy has been shown to perform well when using multi queue. It would be interesting to adapt multi queue scheduling algorithm based on RR in which queues use dynamic quantum values to achieved better reduction in waiting and thus reduce task starvation.

4- Other factors like the energy efficiency, and the power consumption could be taken into account for proper scheduling of tasks.

Arabic Summary

هذه الايام اكتسبت الحوسبه السحابية الكثير من الاهتمامات في العديد من التطبيقات. يستطيع المستخدم ان يستخدم مصادر السحابه بناءاَ على الطلب وفي اي وقت وفي اي زمن. بنية الحوسبه السحابية مناسبه لخدمة عدد كبير من المهام باستخدام المصادر الساحبيه المتاحه. جدولة المهام عامل مهم في الحوسبه السحابية كما انه يقوم بادارة الطلبات الجاهزة للتنفيذ بهدف تحسين السعة الحقيقية للنقل في مصادر الحوسبة السحابية. في الحوسبه السحابية الخاصة، أولوية المستخدم هي واحده من احتياجات المسخدم في المنظمات التي تجب ان تأخذ في عين الأعتبار بحيث تعطى للمستخدمين الذين لديهم طلبات مهمه. على كل حال، الكثير من الباحثين لم يقدموا اي طريقة تامة لحل مشكلة الاستطاله المستحيله الى الآن في الطرق المبنيه على اولوية المهام.

في هذه الرساله اقترحت طريقة فعاله لجدوله المهام على الحوسبة السحابية الخاصة سميت HPJF. الطريقة المقترحة تعين المهام على مصادر السحابة بطريقة مبنية على اولوية المستخدم، وقت المستغرق في التنفيذ، تكلفة التنفيذ، والعبئ على المصدر الافتراضي. بالاضافة الى ذلك تم استخدام تقنيه متعددة الصفوف للحد من مشكلة عدم القدرة على تحصيل المصدر التي تحدث في مثل هذا المنظمات.

تم تطبيق طريقة الجدولة المقترحة باستخدام محاكي يسمى CloudSim. نتائج الطريقة المقترحة تمت مقارنتها مع ثلاث خوارزميات اخرى FCFS، SJF، RR، BLJF. نتائج التجارب والاختبارات تعطي تائج افضل من حيث وقت الانتظار ووقت الوصول مقارنتةُ مع خوارزميات الجدولة الاخرى.

References

Agarwal, D., & Jain, S. (2014). Efficient optimal algorithm of task scheduling in cloud computing environment. *arXiv preprint arXiv:1404.2076*.

Ahmad, E. S., Ahmad, E. I., & Mirdha, E. S. (2017). A Novel Dynamic Priority Based Job Scheduling Approach for Cloud Environment.

Allan, J., Aslam, J., Belkin, N., Buckley, C., Callan, J., Croft, B., . . . Harper, D. J. (2003). *Challenges in information retrieval and language modeling: report of a workshop held at the center for intelligent information retrieval, University of Massachusetts Amherst, September 2002.* Paper presented at the ACM SIGIR Forum.

Anuradha, V., & Sumathi, D. (2014). *A survey on resource allocation strategies in cloud computing.* Paper presented at the Information Communication and Embedded Systems (ICICES), 2014 International Conference on. IEEE; 2014.

Arabnejad, H., & Barbosa, J. G. (2014). A budget constrained scheduling algorithm for workflow applications. *Journal of grid computing, 12*(4), 665-679.

Aslanzadeh, S. (2016). *Anticipatory models of load balancing in cloud computing.*

Awan, M., & Shah, M. A. (2015). A survey on task scheduling algorithms in cloud computing environment. *International Journal of Computer and Information Technology, 4*(2), 441-448.

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern information retrieval* (Vol. 463): ACM press New York.

Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., . . . Hensgen, D. (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), 810-837.

Ababneh, I., 2006. An efficient free-list submesh allocation scheme for twodimensional mesh-connected multicomputers. Journal of Systems and Software 79 (8), 1168–1179.

Chitra, S., Madhusudhanan, B., Sakthidharan, G., & Saravanan, P. (2014). Local minima jump PSO for workflow scheduling in cloud computing environments *Advances in computer science and its applications* (pp. 1225-1234): Springer.

Chugh, C. (2018). A Survey on Several Job Scheduling Techniques in Cloud Computing. *Journal of Network Communications and Emerging Technologies (JNCET) www. jncet. org, 8*(4).

Ergu, D., Kou, G., Peng, Y., Shi, Y., & Shi, Y. (2013). The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment. *The Journal of Supercomputing, 64*(3), 835-848.

Fadhil, M. (2017). Tasks Scheduling in Private Cloud Based on Levels of Users. *International Journal of Open Information Technologies, 5*(4), 22-28.

Fard, H. M., Prodan, R., Barrionuevo, J. J. D., & Fahringer, T. (2012). *A multi-objective approach for workflow scheduling in heterogeneous environments.* Paper presented at the Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012).

Ghanbari, S., & Othman, M. (2012). A priority based job scheduling algorithm in cloud computing. *Procedia Engineering, 50*(1), 778-785.

Ghazizadeh, A. (2012). *Cloud computing benefits and architecture in e-learning.* Paper presented at the Wireless, Mobile and Ubiquitous Technology in Education (WMUTE), 2012 IEEE Seventh International Conference on.

Guo, L., Zhao, S., Shen, S., & Jiang, C. (2012). Task scheduling optimization in cloud computing based on heuristic algorithm. *Journal of networks, 7*(3), 547.

Heinze, T., Ji, Y., Pan, Y., Grueneberger, F. J., Jerzak, Z., & Fetzer, C. (2013). *Elastic Complex Event Processing under Varying Query Load.* Paper presented at the BD3@ VLDB.

Hu, J., Gu, J., Sun, G., & Zhao, T. (2010). *A scheduling strategy on load balancing of virtual machine resources in cloud computing environment.* Paper presented at the Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on.

Javaid, N., Javaid, S., Abdul, W., Ahmed, I., Almogren, A., Alamri, A., & Niaz, I. A. (2017). A hybrid genetic wind driven heuristic optimization algorithm for demand side management in smart grid. *Energies, 10*(3), 319.

Ji, H., Bao, W., & Zhu, X. (2017). Adaptive workflow scheduling for diverse objectives in cloud environments. *Transactions on Emerging Telecommunications Technologies, 28*(2), e2941.

Katyal, M., & Mishra, A. (2014). A comparative study of load balancing algorithms in cloud computing environment. *arXiv preprint arXiv:1403.6918*.

Khajemohammadi, H., Fanian, A., & Gulliver, T. A. (2013). *Fast workflow scheduling for grid computing based on a multi-objective genetic algorithm.* Paper presented at the Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on.

Kroeger, T. M., & Long, D. D. (1999). *The case for efficient file access pattern modeling.* Paper presented at the Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on.

Kumar, P., & Verma, A. (2012). Independent task scheduling in cloud computing by improved genetic algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering, 2*(5).

Kumaresh, V., Prasidh, S., Arjunan, B., Subbhaash, S., & Sandhya, M. (2012). Multilevel Queue-Based Scheduling for Heterogeneous Grid Environment. *International Journal of Computer Science Issues (IJCSI), 9*(6), 245.

LIU, G., & YANG, C. (2013). SCHEDULING RESEARCH BASED ON GENETIC ALGORITHM AND QOS CONSTRAINTS OF CLOUD COMPUTING RESOURCES. *Journal of Theoretical & Applied Information Technology, 51*(1).

Mell, P., & Grance, T. (2010). The NIST definition of cloud computing. *Communications of the ACM, 53*(6), 50.

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing.

Miao, J. (2016). AN ATTRIBUTE-ORIENTED TASK SCHEDULING STRATEGY FOR IMPROVEMENT OF QUALITY OF SERVICE IN CLOUD COMPUTING.

Mohammed, I. (2016). *Task scheduling using best-level-job-first on private cloud computing.* M. Sc. thesis, Middle East University, 2016.(doi not available).

Naseem, M., Al-Rahmawy, M. F., & Rashad, M. Z. (2015). A Scheduling Algorithm to Enhance the Performance and the Cost of Cloud Services. *Computer Engineering and Intelligent Systems, 6*(8).

ROUHI, S., & NEJAD, E. B. (2015). CSO-GA: a new scheduling technique for cloud computing systems based on cat swarm optimization and genetic algorithm. *Cumhuriyet Science Journal, 36*(4), 1672-1685.

Ru, J., & Keung, J. (2013). *An empirical investigation on the simulation of priority and shortest-job-first scheduling for cloud-based software systems.* Paper presented at the Software Engineering Conference (ASWEC), 2013 22nd Australian.

Saini, S., & Kaur, S. (2017). Cloud Computing-An Emerging Technology and Review of Hybrid Models. *International Journal of Engineering and Management Research (IJEMR), 7*(3), 82-85.

Saxena, D., Chauhan, R., & Kait, R. (2016). Dynamic fair priority optimization task scheduling algorithm in cloud computing: concepts and implementations. *International Journal of Computer Network and Information Security, 8*(2), 41.

Selvarani, S., & Sadhasivam, G. S. (2010). *Improved cost-based algorithm for task scheduling in cloud computing.* Paper presented at the Computational intelligence and computing research (iccic), 2010 ieee international conference on.

Sotomayor, B., Montero, R. S., Llorente, I. M., & Foster, I. (2009). Virtual infrastructure management in private and hybrid clouds. *Internet computing, IEEE, 13*(5), 14-22.

Topcuoglu, H., Hariri, S., & Wu, M.-Y. (1999). *Task scheduling algorithms for heterogeneous processors.* Paper presented at the Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth.

Wyld, D. C. (2009). *Moving to the cloud: An introduction to cloud computing in government*: IBM Center for the Business of Government.

Xu, H., Yang, B., Qi, W., & Ahene, E. (2016). A multi-objective optimization approach to workflow scheduling in clouds considering fault recovery. *KSII Transactions on Internet and Information Systems (TIIS), 10*(3), 976-995.

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

Singh, A. K., Sahu, S., Gautam, K. K., & Tiwari, M. N. (2014). Private cloud scheduling with SJF, bound waiting, priority and load balancing. International Journal, 4(1).

X. Meng, V. Pappas and Z. Li, " *Improving the scalability of data center networks with traffic-aware virtual machine placement*.," In INFOCOM, 2010 Proceedings IEEE, pp. 1-9, 2010.

Guang Liu, Chen Yang and Daoguoli. "*Scheduling research based on genetic algorithm and Qos constraints of Cloud Computing*". Journal of Theoretical and Applied Information Technology, 10th May 2013. Vol. 51 No.1, pp. 92–95.

S. Kaisler, W. H. Money, and S. J. Cohen, “A decision framework for cloud computing,” Proceeding IEEE 45th Hawaii Int. Conf. Syst. Sci. A, pp. 1553–1562, 2012.

Zhu, X., He, C., Li, K., & Qin, X. (2012). Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters. *Journal of parallel and distributed computing*, *72*(6), 751-763.

Ababneh, I., & Bani-Mohammad, S. (2011). A new window-based job scheduling scheme for 2D mesh multicomputers. *Simulation Modelling Practice and Theory*, *19*(1), 482-493.

Babbar, D., & Krueger, P. (1994, August). On-line hard real-time scheduling of parallel tasks on partitionable multiprocessors. In *1994 Internatonal Conference on Parallel Processing Vol. 2* (Vol. 2, pp. 29-38). IEEE.

Wu, Z., Ni, Z., Gu, L., & Liu, X. (2010, December). A revised discrete particle swarm optimization for cloud workflow scheduling. In 2010 International Conference on Computational Intelligence and Security (pp. 184-188). IEEE.

Buyya, R., Broberg, J., & Goscinski, A. M. (Eds.). (2010). Cloud computing: Principles and paradigms (Vol. 87). John Wiley & Sons.

Ahmed, A., & Sabyasachi, A. S. (2014, February). Cloud computing simulators: A detailed survey and future direction. In 2014 IEEE International Advance Computing Conference (IACC) (pp. 866-872). IEEE.