I.J. Modern Education and Computer Science, **2018, 2, 1-15**
Published Online February 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2018.02.01

MLRTS: Multi-Level Real-Time Scheduling Algorithm for Load Balancing in Fog Computing Environment

Mohamed A. Elsharkawey[1]
Email: melshrkawey1964@yahoo.com

Hosam E. Refaat[2]
[1, 2] Suez Canal University, Faculty of Computers & Informatics, Information System
Department Ismailia 41522, Egypt
Email: hosam.refaat@ci.suez.edu.eg

Received: 09 October 2017; Accepted: 29 November 2017; Published: 08 February 2018

***Abstract*—**Cloud computing is an innovative technology which is based on the internet to preserve large applications. It is warehoused as a shared data over one platform. In addition, it offers better services to clients who belong to different organizations. In spite of the maximum utilization of computational resources provided by the cloud computing with lower cost, it suffers from specific restrictions. These restrictions are encountered through the load balancing of data in the cloud data centers. These restrictions are represented in the less bandwidth utilization, resource limitations, fault tolerance and security etc. In order to overcome these limitations, new computing model called Fog Computing is presented. It aims to offer the required service of the sensitive data to end users without delaying. The function of the fog computing is similar to the cloud computing with two preferred advantages. The first one is that it is placed more near to the end users to introduce its service in less time. Secondly, it is more valuable for streaming the real time applications, sensor networks, IOT which need high speed and reliable internet connection.

In this paper, a novel load balancing algorithm has been proposed over a novel architectural model in the Fog Computing environment. The proposed model aims to serve the real-time tasks within their deadline. In addition, it serves the different soft tasks without starving. The soft tasks are classified according to the execution time and the priority levels. In addition, they are served according to their waiting time and priority-level. Furthermore, the proposed algorithm is employed to maximize the throughput, the resources and the network utilization and preserving the data consistency with less complexity to accomplish the end users demand.

***Index Terms*—**Cloud system, Fog computing, resource allocation, Real-Time Systems.

I. Introduction

Cloud computing is introduced as a recent technology, which is utterly reliant on the internet. The architecture of the cloud computing is based on a central server that maintain a huge amount of sharing database, different resources and a large number of commercial applications. On the other hand, an enormous number of remote clients that belongs to different organizations can benefit from the different services provided by the central server as shown in Figure 1. Each remote client has its own, operating system and web browser that work independently on the contents of the cloud server [1, 2]. The connection of the client to the internet is the only requirement from the client to utilize the cloud server abilities. So, the IT industry and any small organization can acquire these services from the cloud center without spending huge amount of money in hardware or software.

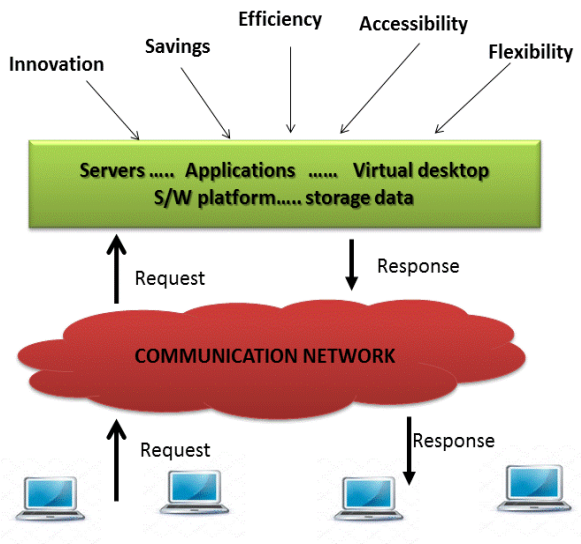

Fig.1. Cloud Computing Features

Actually, the implementation of the cloud presents several related concepts. These concepts deal with virtualization, resource allocation, computing distribution, utilization of bandwidth, load balancing, fault tolerance, high availability and dynamic scalability for different categories of data and applications. The management of the operations related to all these concepts is performed by the cloud service provider.

Virtualization is the one of the major challenging in the cloud computing to accomplish the optimum utilization of the cloud resources. The virtualization is implemented by allocating numerous virtual machines (VMs) on the individual physical server known as host server. However, each virtual server works as a physical server which multiplies the capacity of any single physical machine. Each virtual machine has its own applications and its own resources. The running processes are distributed on the Virtual machines in parallel manner. Therefore, the execution of each process is accomplished in less time through the magic of virtualization [3].

The cloud providers assign the resources to the end users as a service depending on the dissimilarity of the service models and also based on the user requirements. The service models may include Software as a service known as SAAS, Platform as a service known as PAAS, Infrastructure as a service known as IAAS. These services are slanted on each other and in a pool way.

Generally, the implementations of the different processes on the cloud present several benefits to the end users. At First, the data becomes shared over one platform, so better services are delivered to each client. Secondly, the end user can get his on-demand services or resources usage in a secure, reliable and flexible manner according to his need only.

Despite of these benefits that can be offered by cloud computing to massive applications, it suffers from a set of certain restrictions [4]. The first restriction occurs when the numbers of the end users are increased. In this case, the demands are enlarged to get more services than the cloud capacities. This increases leads to the high latency of the accessible services unless the available resources and the available bandwidth are upraised to acquire all the extra requests. The second limitation occurs when the data produced by the cloud service is transferred through a long distance from the cloud center to the end users. The far distance may affect the data security and maximize the probability of losing. Furthermore, an irregular excess in the workload may cause the need to validate a novel load balancing technique. The load balancing is the fair allocation of the work load among multiple computing resources such as networking, hard drives and computers [5]. So, it will be required to accomplish the development in the employment of the computation resources and storage devices.

In order to overcome these restrictions, a new technology of highly virtualized computing model has been presented known as Fog computing. The model [6] is proposed by CISCO to be held as cloud edge of an enterprise network. The occupation of the fog computing is not a replacement of the cloud computing. Actually, it works as a supportive environment that has the ability to provide high QoS to the different client requests of the near distances. So, the whole fog-cloud environment consists of a set of fogs computing servers and a set of the clouds computing centers as shown in Figure 2.

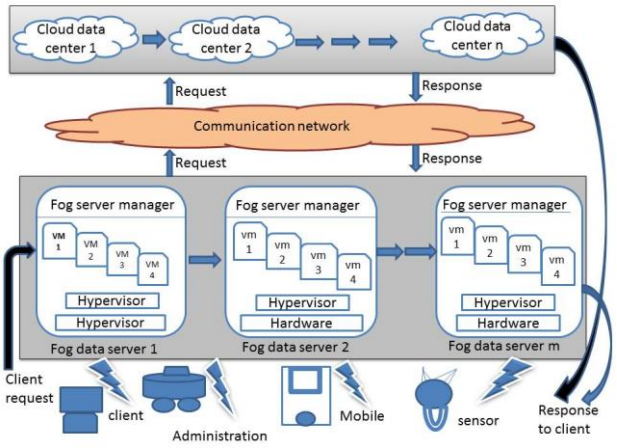

Fig.2. Fog Computing Features

Generally, the operations of the fog computing is similar to the cloud computing with two main differences. The first difference is related to the location of the fog computing that is placed very close to the end users. Hence, the fog computing can be imagined as a local cloud. The second difference deals with the resources abilities of the fogs that have fewer abilities compared to the abilities of the cloud resources. However, each fog computing include its own server that is supported by its own resources. In addition, each fog server is occupied by the necessary software or firmware to establish the required VMS such as the hypervisor.

From the point of the interaction view, the location of the fog computing seems as an intermediate layer between the end users and the cloud computing data centers presented within the internet. Actually, the fog computing presents massive features for storage devices, computation resources and networking. Therefore, the fog computing is more appropriate for the applications that may need real time service, low latency and the geographical distribution support. So, the fog computing offers great aids in numerous areas of business process optimization, agriculture, deep sea survey, health manufacturing, merchandise prices, real time intelligence, smart orders, weather forecasting and many other areas.

Consequently, the fog computing has to be able to deal with the devices that are operable at a very rapid rate such as Internet of things (IOTs) [7]. IOTs are basically a network of wireless things including ordinary devices from medical devices to home machines.

The emerging of the fog computing presents a novel prearrangement in the on-demand service provided to the end users. When the fog computing is implemented the on-demand services may provide by one of three methods. At First, the services may be offered by the closest fog computing. Secondly, they may be offered by one of the neighboring fogs computing closest to the demanded fog

within a specified region. Finally, they may be offered by the cloud computing communicated to this specified region. The election of which service provider will be used to perform the requested task is based on three aspects. The first one is the type of service required to be served that is either a service of real time or service of non-real time. The Second one is the amount of the waited time for current work load that is needed for the services in the intended fog and its neighboring ones within a specified region and the related cloud. The third aspect deals with the used strategy of the work load distribution to accomplish the required load balance.

In general, the Load balancing appears to play a vital role for scheduling the different types of the users' tasks. Load balancing can be classified [8] into different fundamentals such as the applied state that maybe static or dynamic, the load balancer type which is hardware or software and the policies rules such as resource, information, selection, location and transfer. In the fog-cloud environment, the load balance must be accomplished in two situations. The first situation is between the end users and fog layer i.e. at the intermediate layer. The second situation is between the fog and cloud layer. So, the job scheduling algorithm is used to allocate the load from the clients to all servers to satisfy the fair distribution. The achievement of the fairness will minimize the long time waiting of any task. In addition, it will increase the execution speed of the user's tasks in using the available resources with optimum consumption of storage to minimize the response time of the submitted tasks.

In fact, the load balancing includes two basic approaches. They are the static load balancing approach and the dynamic load balancing approach. In the Static load balancing approach, the decisions are made in deterministic way during the run time. The decisions are based on the performance of the processing nodes that remained unchanged during the run time. Also, the number of tasks in each node is unchanged [9]. The methods of the static load balancing are non-preemptive i.e. the allocated load to the node cannot be moved to another node. On the other hand, the dynamic load balancing decisions are taken during the execution based on the states of the information at the run time [10]. The dynamic load balancing algorithms redistribute the work load is based on the changes in all the workloads which are monitored through the working of the system.

In the following, the rest of the paper is organized as follows. Section II; discuss the related work of the load balancing algorithms and techniques that are proposed for working with the cloud systems. In section III, the architecture of the proposed model is presented. In addition, the details of the modules that are included within the model are clarified and explained. In section IV, the performance evaluation and the results of the simulations are introduced. Section V concludes the paper and provides the venues for the future work.

II. Related Work

In this section, Several Load Balancing algorithms are introduced for various authors. These algorithms are studied and reviewed based on the different available parameters such as deadline, execution time, bandwidth, cost, priority, reliability, scalability, task length and throughput. Essentially, the efficient load balance algorithms have been implemented in the cloud technologies. In this paper, the proposed algorithm is introduced to be implemented in the fogs computing environment. However, this illumination is revealed according to the relation between them.

Generally, the management of the load balancing operations is similar in both of the cloud and fog computing with only main difference. In the fog computing, the load balancing makes the balancing operation more feasible and effective with the limited resources. It offers access to the resources of less bandwidth and time. So, the Fog computing has satisfied the needs for the nearest end users at a tremendous rate without any confusion similar to what may occur for the network traffic.

In this section, the first load balancing technique is introduced in [11]. This technique is designed to accomplish good services by increasing the resource utilization based on two parameters, which are the task priority and its length. The selection of the tasks for the scheduling may be obtained from both of the first and last indexed queue to achieve a more steady system. The tasks are scheduled based on the total credit system sponsored from grouping of credit length computed from task length and credit priority computed from the task priority. Finally, the priority of execution is given to the high credit task. However, this algorithm suffers from certain weaknesses when the total credits of several tasks became identical. In this case, the FCFS has to be implemented without guarantee of tasks to be completed earlier or to its deadline.

Another algorithm is based on analogous behavior of Honey bee model (HBB-LB) is proposed by Dhinesh babu L.D et al. [12]. In this algorithm, the priority is taken as a main QoS factor to Bar any process from waiting for a long time in the queue to reduce the execution time and maximize the throughput. HBB-LB algorithm depends on two types of bees. The first type is known as the scout bees. Its role is searching for the food source until it is found. So, the second type is defined as forager bees receive a signal from the scout bees. This signal will carry the required information about the quantity, the quality, and the distance from the beehive through the waggle/tremble/vibration dance. However, when the signal is strong, it means more available foods. Thus, the forager bees will track the short path determined by the scout bees to get the food location. When the forager bees get their required foods from the source, they work as the scout bees to inform the other bees about how much food is still left and so on.

In the same way, the tasks can be represented as the Honey bees and the Virtual Machines can be represented as food sources. In addition, The VMs are categorized according to three situations, balanced overload, high overload and low overload. When the VMs are overloaded, the tasks are removed and act as a honey bee. So, these tasks are submitted to the VMs that has the low overload. These assignments are reliant on how many high priority tasks are performed on those VMs. It must be noted that the selection of the VM is performed only for the VM which has the low overload and the least number of the executed priority tasks. After appropriate assignment of tasks on VM, all information is updated so that the remaining tasks can obtain their needs under load VM. This algorithm has introduced certain advantages represented in the appropriate resource utilization; maximizing the throughput while keeping the other QOS parameters which are built on the task priority. On the other hand, the drawbacks are presented for the low priority tasks which suffer from idle state or long time waiting in the queue. These tasks may be neglected causing the unbalancing of the workload balancing.

The dynamic and the optimization of the centralized based algorithm presented for the load balancing in [13]. In this algorithm, the decision of distribution is taken by the central node. The decision is based on the workload of fewer messages. However, the shortcoming is occurred when the central node fails. In this case, the entire work of the system will be stopped causing the corruption of the system performance. So, better performance can be accomplished by obtaining the maximum throughput based on the optimization which is considered as one of the possible solutions. It can be performed in two ways. The first way implements the method that is defined as the complete method. So, the valid values are allocated to all variables to get the intended results. If one of the allocated values gets to be incorrect the solution is excluded. The second way is defined as the incomplete solution and the key factor is used as a probability. It assumes that its solution based on the input parameters which give more correct answers. The characteristics of these parameters have to offer the simplicity, effectiveness, and speed for resolving problems. This approach is known as Stochastic Hill Climbing. It is the most preferred one to solve the optimization problem. The Multi-Objective tasks scheduling algorithm that is based on the offering of an efficient resource utilization to enhance the throughput is introduced in [14].

This algorithm accomplishes the decrease of the cost of an application running in a SAAS environment without changing in the service level agreements. In this algorithm, the tasks are tied to the VMs by a way that achieves faster execution. The algorithm is applied based on two main steps. At first, the priorities are assigned to the tasks such that the High QoS is set to the low value and the low QoS is set to the high value. Hence, the tasks of lower values has the higher priorities and vice versa. Secondly, the QoS values are allocated to the VMs such that the high QoS values are allocated to VMs that have high MIPS and low QoS values are allocated to VMs that have low MIPS. On the other hand, the sorting function is performed to arrange the tasks based on the minimum size and minimum QoS value. The sorting function is implemented in descending order from the high MIPS to the low MIPS. After the completion of sorting, the tasks are allocated to a list of the sorted VMs. The allocation is performed such that the first VM in the VM list is assigned by the first task in the task list. By the same way, the second VM in the VM list is allocated by the second task in the task list. In addition, the allocation process proceeds in the same way for the further tasks with the remaining VMs. When all the VMs in the VMs list are occupied by all the tasks, the upcoming tasks is allocated to the first VM and this process continues. In this algorithm, the limitations are introduced due to the usage of the minimal QoS parameters such as the execution time. So in the future, it requires the other QoS factors to be added.

An Optimal Model for priority based service Scheduling Policy is introduced in [15]. This algorithm is based on the priority and the admission control as a service scheduling policy to offer the moral optimization and the maximum throughput. The algorithm operations are based on the offering of the fully utilization of the available resources. It aims to serve the user requests with less time spent in the queue. However, the higher precedence in using the cloud services is presented to the user that pays higher than the others. In this algorithm the performance is affected by the limitations of the applied features. Especially, the futures that are related to the security and the resources that hired from other cloud.

In order to manage the massive data of the expensive load balancing for the physical network, the framework for the virtualization network load balancing has been proposed in [16]. In this context, the load balancer in the data center is organized to be adjusted dynamically according to the change in the data of each customer requirement and the availability of the network service providers. The balancing is applied in the two levels of the master and the slave. The first load performs as the master where the other performs as a slave that will be selected by the load balancer and network load balancer in the future. High availability of web services is obtained due to the employment of this network load balancer. In this framework, the employment of software-type load balancer is favored than hardware-type load balancer that adds lots of efforts and financial burdens on the users. Generally, some of the advantages have been introduced as follow. In the single network, the web connection limitation has been resolved. In addition, the implanted algorithm can be updated through the users by changing the number of the load balancers. In the future, the performance may be enhanced for a huge number of the users participated on the internet by applying it in the hybrid cloud.

Actually, the various scheduling algorithms of the changing in the used QoS parameters have been offered for different environments in [17]. The scheduling is performed to accomplish the massive income and to improve the effectiveness of the work load. Therefore,

there are several versions that are implemented for each of the dissimilar types of the scheduling algorithms such as FCFS algorithm, Min-Min algorithm and Max-Min algorithm, Round-Robin algorithm. However, the most effective one among them is the heuristic technique. Its scheduling operations involve three phases in a cloud computing environment. At first, the resources are located. Then, the best target resource is selected. Finally, the task is submitted to the target resource. Recently, the real efficient time scheduling (RETS) is introduced in [18]. It aims to verify the executing the real-time tasks without delay. So, it saves ten percentages from the all available resources for the real time tasks. However, this percentage is lost if there are no real-time tasks.

In the end of this section, it is important to mention that the Scheduling objective is to reduce the response time and completely exploiting the resources. So, different scheduling algorithms have been proposed based on the deadline. For these algorithms the selection of the task by different tools and over dissimilar environment has been compared in [19]. These algorithms are developed from the different viewpoints such as execution time, cost, delay, response time and resource utilization time.

In general, various types of the scheduling algorithms have been offered based on the deadline such as the sporadic task approach with the deadline constraints, the Preemptive scheduling of online real-time service with the migration of the task , the Priority and the admission control based algorithm, the schedule–as–soon-as-possible algorithm and the level based scheduling algorithm.

III. Proposed Model

Generally, the fog computing servers are spread in the neighboring manner. Each fog computing server is centered in the specific location mainly to serve all the clients requests in a specific region. So, each fog server is supplied by its own load balancing algorithm. The proposed model is designed to serve the different real-time tasks or the soft-tasks required by all clients in the fog computing region. In addition, it is introduced the required services for all the soft tasks that may arrive from any neighboring fog. This case will be occurred when one of the neighboring fogs is suffered from the excess load of the soft tasks. Furthermore, the proposed load balancing algorithm has been applied to satisfy the tasks issues. So, it deals with the deadlines time, the execution time, the consistency of data and appropriate resource utilization.

The architecture of the proposed load balancing model in the fog computing is shown in Figure 3. It consists of four main modules. They are the Classifier, Task-load Monitor, (TLM), the Fog-Cloud-Balancer (FCB) and the VM-Manager (VMM). In the following, subsections, the design of the proposed model for the load balancing will be introduced and explained.

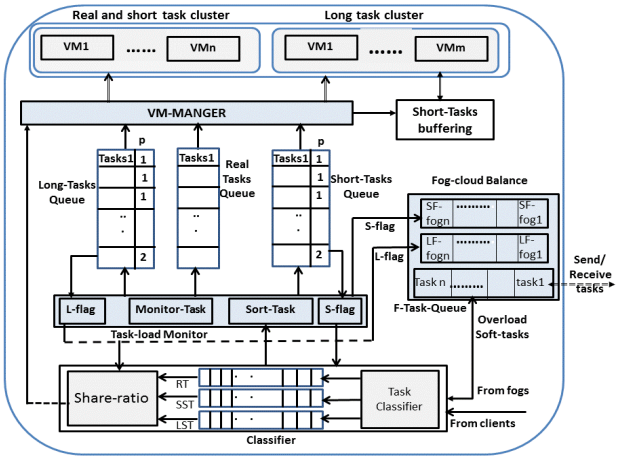

Fig.3. The Proposed model for Fog Data Center.

A. The Classifier

The Classifier module includes two basic functions. They are the Task-Classifier and the Share-Ratio function. In the following, these functions are explained.

1. The Task Classifier

The Task-Classifier is used to differentiate between the received tasks that may be of the real-time type or the soft type. However, the soft tasks are classified again into two different types. They are the Long-Task and the Short-Task. This classification is performed based on the expected execution time of each task compared with the defined threshold value that is defined as (Ω). The soft task of the expected execution time less than the threshold value is known as the Short-Task. In addition, the soft task of the expected execution time greater than the threshold value (Ω) is defined as the Long-Task. The threshold value is a dynamic value that may be determined and changed by the system administrator based on the nature of the service provided. According to these classifications, the received tasks are differentiated within the Task-Classifier before submitting to the (TLM). Each type of these tasks will be inserted and sorted in its own queue through the (TLM). However, the execution times of the real-time tasks are stored in the RT queue. The execution times of the Short-Tasks are stored in the SST queue. The execution times of the Long-Tasks are stored in the LST queue.

The operations of the task classifier are performed according to the values of the flags in the TLM. These flags are defined as the S-flag and the L-flag. The two flag types will be introduced and explained in the section of the TLM. In addition, the type of the incoming task will affect the execution operations of the Task-Classifier.

The whole operations of the two different methods in the Task-Classifier can be clarified in the following procedures.

Task-Classifier function

Step 1: Classify the received task type (real or soft)
Step 2:
- If ((task is real) and (S-flag=1) or (L-flag=1))
  - Compute its expected execution time

- Save the value of the expected execution time in the RT queue
- Submitted the task to the Task-Monitor to be inserted in the Real-Tasks Queue.

- Else If ((task is real) and (S-flag=0) and (L-flag=0)):
  - Submitted the task to the Monitor-Task to be inserted in the Real-Tasks Queue.

Step 3: If the task is soft:
- Compute its expected execution time.

Step 4: Classify the type of the soft task:
- Compare the expected execution time of each soft task with the threshold value (**Ω**).

Step 5: If ((the expected execution time of the soft task is > **Ω**) and (L-Flag=0) and (S-flag=0))
  - Submitted the Long-task to the Monitor-Task to be inserted in the Long_Tasks Queue

- Else If ((the expected execution time of the soft task is < **Ω**) and (S-Flag=0) and (L-flag=0))
  - Submitted the Short-task to the Monitor-Task to be inserted in the Short_Tasks Queue

Step 6: If ((the expected execution time of the soft task is > **Ω**) and (L-Flag=0) and S-flag=1))
  - Submitted the Long-Task to be inserted in the Long-Task-Queue
  - Save the value of the expected execution time in the LST queue

Step 7: If ((the expected execution time of the soft task is > **Ω**) and (L-Flag=1) and S-flag=0))
  - Submitted the Long-Task to the FCB to be inserted in the F-Task-Queue
  - Save the value of the expected execution time in the LST queue

Step 8: If ((the expected execution time of the soft task is < **Ω**) and (S-Flag=0) and L-flag=1))
  - Submitted the Short-Task to be inserted in the Short -Task-Queue
  - Save the value of the expected execution time in the SST queue

Step 9: If ((the expected execution time of the soft task is <**Ω**) and (S-Flag=1) and (L-flag=0))
  - Submitted the Short-Task to the FCB to be inserted in the F-Task-Queue
  - Save the value of the expected execution time in the SST queue

Step 10: the expected execution time of the soft task is <**Ω**) and (S-Flag=1) and (L-flag=1))
  - Submitted the Short-Task to the FCB to be inserted in the F-Task-Queue
  - Save the value of the expected execution time in the SST queue

Step 11: the expected execution time of the soft task is
- > **Ω**) and (S-Flag=1) and (L-flag=1))
  - Submitted the Long-Task to the FCB to be inserted in the F-Task-Queue
  - Save the value of the expected execution time in the LST queue

Step 12: IF ((L-flag turn to 0) while (S-flag=0)) 7

$$\sigma_{deff} = \sum_{i=1}^{m} ExeTime(Long-Tasks) - \sum_{j=1}^{n} ExeTime(Short-Tasks) + \sum_{j=1}^{n} ExeTime(\text{Re}al-Time-Tasks)$$

  - Call Share-Ratio(σ_{deff})

Step 13: IF ((S-flag turn to 0) while (L-flag=0))

$$\sigma_{deff} = \sum_{i=1}^{m} ExeTime(Long-Tasks) - \sum_{j=1}^{n} ExeTime(Short-Tasks) + \sum_{j=1}^{n} ExeTime(\text{Re}al-Time-Tasks)$$

  - Call Share-Ratio(σ_{deff})

Step 14: IF ((L-flag turn to 0) and (S-flag turn to 0))

$$\sigma_{deff} = \sum_{i=1}^{m} ExeTime(Long-Tasks) - \sum_{j=1}^{n} ExeTime(Short-Tasks) + \sum_{j=1}^{n} ExeTime(\text{Re}al-Time-Tasks)$$

  - Call Share-Ratio(σ_{deff})

2. Share- Ratio function

According to the operation performed in the classifier, the Share- Ratio function is presented. It aims to compute the resources ratio that should be assigned to each type of tasks. The Share-ratio for each type of the tasks is computed based on the total amount of the execution-time of each type. The procedure of the Share-Ratio computation can be clarified in two principle cases. The initial case is pushed at the starting of the system operations. Secondly, the iterative case that will be implemented according to the different iterations determined and called by the Task-classifier. Generally, the VMs are divided between two clusters. Each cluster comprises the half of all the fog computing resources. So, the maximum waiting times in both clusters, defined as follows:

σ_L : defined as the maximum allowable waiting time for the last task that may be appended to the Long-Task-Queue.
σ_{SR} : defined as the maximum allowable waiting time for the last task that may be appended to the Short-Task-Queue .

For initial state, the values of σ_L and σ_{sr} are equal, and both of the values are assigned to the value of (multiple number of VM in each cluster * 1 sec)

On the other hand, the iterative case is defined as the ordinary case of the system. The value of the share ratio of each cluster is recomputed according to the change in the status value of the S-flag and L-flag that are based on the Task Load Monitor. In addition, the values of the maximum allowable waiting times σ_{SR} and σ_l are recomputed according to the execution times of the different types of all the tasks. In the following, the share ratio re-computation through the iteration case can be explained as follows:

Step 1: if ($\sigma_{deff} > 0$)

$$\sigma_L = \sigma'_L + \sigma_{deff}$$

$$\sigma_{SR} = \sigma'_{SR} - \sigma_{deff}$$

Else

$$\sigma_{SR} = \sigma'_{SR} + \sigma_{deff}$$

$$\sigma_L = \sigma'_L - \sigma_{deff}$$

Where:

σ'_L : is the total sum of the remaining execution time of the tasks in Long-Task-Queue.

σ'_{SR} : is the total sum of the remaining execution time of the tasks in the Short-Task-Queue and the Real-Task-Queue. Hence, the share ratio of the reserved resource to serve the VMs in each cluster is computed according to the following equation.

$$C_L = \frac{\sigma_L}{\sigma_L + \sigma_{SR}}$$

$$VM_L = \lfloor N * C_L \rfloor$$

$$VM_{SR} = N - VM_L$$

Where:

C_L : The ratio of the resource allocated to serve the Long-Tasks.

N: The maximum number of the VMs that can be created by the fog server.

VM_L : The total number of VMs that will be assigned to the Long-task-Cluster.

VM_{SR} : The total number of VMs that will be assigned to the Real and short-task-Cluster

Step 3: notify the VM-Manager by the computed ratios to use them in allocating the VMs for each cluster.

B. Task-Load-Monitor (TLM)

The TLM comprises four components. The first two components are the flags. They are defined as the S-Flag and the L-Flag. Each of them is used to indicate the status of load for a type of the soft tasks. The S-Flag is used to refer to the status of load in the queue that contains the Short-Task. The L-Flag is used to refer to the status of the load in the queue that contains the Long-Tasks. Set the value 0 to any of the both flags, mean its queue in the local fog is able to receive additional task. Otherwise, set the value 1 to any of the both flag, which mean its queue becomes fully loaded. Each soft task belongs to a flag of the value =1 will be directed by the Task-classifier to the F-Task-Queue.

However, the other components represent the two functions that are performed in the TLM. They are defined as the Sort-Task function and the Monitor-Task function. In Fact, the operation management of the TLM components depends on the status of the soft loads in their queues. The status of the soft loads can be assembled into two different cases in the proposed model. The first case is encountered when the Share-Ratio is re-computed. In this case, the Monitor-Task function is halted.

On other hand, the procedures of the Sort-Task function are synchronous in the execution with the Share-Ratio computation in the classifier. The Sort-Task function can be summarized in inserting and sorting each task according to each type in its own queue. The real tasks are sorted according to their deadline time. The set of the soft tasks that are sorted is performed according to the SJF scheduling. In addition, the value of the priority-level=1 is assigned to the all tasks in each soft queue when the Share-Ratio is recomputed. The similar priorities =1 for these tasks mean the sorted tasks at the priority-level =1 will remain unchanged until they are executed by the VMs.

On other hand, any of the soft tasks may arrive after the Share-Ratio is recomputed and will be assigned to the priority-level =2.

However, all the tasks are considered after the Share-Ratio computation were loaded to their queues, the values of the S-Flag, L-Flag for this state are defined as follows:

- S-Flag = 0
- L-Flag = 0

This case is defined as the reasonable state that should be obtained after the Share-Ratio computation. In addition, both values of the σ_L and σ_{sr} are re-defined as follows:

- σ_L: is evaluated to be the value of the waiting time of the last task in the long-Task queue.

This means the waiting time for any Long-Task should not exceed the value of the σ_L until the Share-Ratio is re-computed again.

- σ_{SR}: is evaluated to be the value of the waiting time of the last task in the Short-Task queue.

This means, the waiting time for any Short-Task should not exceed the value of the σ_L until the Share-Ratio is re-computed again.

The σ_L and σ_{sr} are used as a constraint values to control the load balancing and the module operations of the proposed architecture in the fog computing. The S-flag value will be remained = 0 as the waiting time of the last appended task to the Short-Task-Queue is less than σ_{sr}. Similarly, the L-flag value will be remained = 0 as the waiting time of the last appended task to the Long-Task-Queue is less than σ_L.

During the reasonable state, some tasks of the different queues types are executed on their allocated VMs. On other hand, some of the new tasks are reached to these queues after obeying to the Task-Classifier operations. Therefore, the Sort-Task function is halted while the Monitor-Task is pushed to accomplish three main objectives. At first, it inserts and sorts each task in its own queue. Then, it assign a priority-level = 2 to each upcoming soft tasks when inserted in their soft queues of

the local fogs. Finally, it continuously observes the waiting time of each soft task appended to the tail in both of the soft queues.

On other hand, the real-time tasks are inserted in the Real-Tasks-Queue. They are sorted according to their deadline time. However, they are pushed to be executed according to their deadline time only. The execution is done regardless the state of the soft queues in the system.

1. Sort-Task function (during Share-Ratio computation)

Step 1:
- Sort the Short-Tasks in the Short-Task-Queue according to SJF
- Assign the value 1 to the Short-Tasks priority.
- Sort the Long-Tasks in the Long-Task-Queue according to SJF
- Assign the value 1 to the Long-Tasks priority.
- Sort the Real-Tasks according to their deadline time in the Real-Tasks-Queue
- Add the execution times of the Real-Tasks to the waiting time of the last task in the Short-Task-Queue.

Step 2:
- Set the values of the S-Flag = 0 and L-Flag = 0.

2. Monitor-Task function

Step 1: For each arrived task:
- If real-time task, insert it in the Real-Task-Queue
- Add the execution time of this Real-Task to the waiting time of the last task in the Short-Task-Queue.
- If soft task, compute its waiting time

Step 2:
- If the waiting time for the last Long-Task < $\sigma_{L.}$, assign the value 2 to the priority-level of that task.
- Re-sort all the tasks of the priority-level =2 according to SJF schedule.
- If top task has priority-level =2. (i.e. the task becomes occupying the top of the queue and all tasks has the priority-level=1 are transferred to the VMs)
  - Stop the SJF sorting of the tasks that has priority-level =2.
  - Change the priorities of all the currently existing tasks in the queue to level =1
  - For each upcoming task in the queue, set its priority-level to 2.

Step 3: If the waiting time for the last long task > $\sigma_{L.}$
- set L-Flag =1
- Stop receiving tasks form the Task-Classifier.
- Change the priority-level of all the tasks existing in the Long-Task-Queue into the priority-level =1

Step 4: notify the tasks classifier to send the soft Long-Tasks to the FCB.

Step 5: notify the Task-Classifier by new L-Flag =1

When the L-Flag or S-Flag or both change from 0 to 1, the model is transferred from the reasonable state to other state defined as an effective state. This state means the system is heavily overloaded. Hence, the incoming tasks that have the flag value =1 will be directed to the F-Cloud Balancer to be executed. In addition, the Share-Ratio should be re-computed before backing to the reasonable state. Furthermore, the excess load should be performed before the Share-Ratio is re-computed

Notice: the same procedure steps (2:5) that applied to Long-Task-Queue are applied to the Short-Task-Queue with reference to σ_{SR}

C. The Fog-Cloud Balancer (FCB)

The third module is defined as the Fog-Cloud-Balancer (FCB). It consists of the two vectors that defined as the SF-Fog and the LF-Fog. Each vector of them comprises the number of slots that equal to the number of all the neighboring fogs in addition to the local fog. Each slot in the SF-Fog vector is allocated to one of the Short-Tasks queues in the fogs computing. Each slot is used to refer to the loads circumstances of the Short-Tasks queues in the definite fog computing. When the slot value is set to 1, this means that the queue of the Short-Tasks of that fog is fully loaded. Otherwise, the slot value is set to 0 if the queue of the Short-Tasks able to receive additional tasks. By the same manner, each slot is used to refer to the loads circumstances of the Long-Tasks queues in a definite fog computing. When the slot value is set to 1, this means that the queue of the Long-Tasks of that fog is fully loaded. Otherwise, the slot value is set to 0 if the queue of the Long-Tasks is able to receive additional tasks.

Based on the circumstances of all the slots in both of the vectors, each fog computing is alerted by the loads circumstances for both of the types of the soft queues in all the fogs.

Actually, the main FCB function is presented when the value of the S-Flag or L-Flag of local queues is changed to 1. This means that the local queue of the Short-Task or Long-Task in turn is fully loaded.

So, the coming Short-Tasks or Long-Tasks are delivered to the FCB from the Task-Classifier. In this case, the FCB guided the received tasks either to one of the neighboring fogs or to the cloud computing. The following steps clarify the operations of the FCB. Actually, the main function of the FCB is activated when the value of the S-Flag or the L-Flag in the local queues is changed to 1. In this case, the local queue of the Short-Task or the Long-Task in turn was fully loaded. So, the coming Short-Tasks or the Long-Tasks are delivered to the FCB from the Task-Classifier. Accordingly, the FCB guides the received tasks either to one of the neighboring fogs or to the cloud computing. The following steps clarify the operations of the FCB.

Step 1: If S-flag or L-Flag is set to 1, broadcasting all the fogs with the new local Flag status.

Step 2: Change the value of SF-Fog or LF-Fog according to the up-to-date status in other fogs.

Step 3: FCB receives the tasks from the Task-classifier If S-flag =1 or L-Flag =1

Step 4: According to the type of soft task (long or short), FCB finds the nearest fog based on the value of its dedicated flag slot in the flag vector (SF-Fog or LF-Fog).

Step 5: If there is a free fog in the flag vector (i.e. it has zero value in SF-Fog or LF-Fog), FCB send the task to that fog for execution.

Step 6: If there are no free fogs found to execute the task, FCB send it to the related cloud.

D. The VM-Manager (VMM)

The fourth module is the VM-Manager (VMM) that consists of a set of the homogenous virtual machines. The role of the VMM can be clarified through two main functions. The first function is dividing the whole VMs into a two main clusters. The first cluster comprises the VMs that are allocated to the both of the real-time tasks and the short-soft tasks together. The second cluster includes the VMs that are allocated to the long-soft tasks only. This division is performed to allow the existing load of the tasks in each cluster to obtain an appropriate Share-Ratio from all the existing VMs. So, the Share-Ratio is computed for each cluster based on the existing loads type in each cluster relative to all the existing load types in both of the clusters.

The Share-Ratio is not a constant value. It is changed according to the effective change (defined as an effective state in the TLM) in the existing loads type of each cluster. So, the change in the Share-Ratio (number of assigned VMs) to each cluster should be performed.

This change allows each cluster to obtain an appropriate new Share-Ratio from all the VMs of both clusters.

However, the instantaneous migration of the VMs from one cluster to another may cause a problem if they are busy by executing tasks. So, the instantaneous migration may be performed only if enough numbers of the free VMs are available. On the other hand, if the required numbers of the VMs are not available, the VMs migrations may be postponed until the VMs ends their current tasks and become free.

The second function of the VM-Manager is the allocation of the tasks on the VMs. For both of the types of the soft tasks, the tasks are managed as a two dissimilar groups. The first group is directly inserted after the Share-Ratio computation. The value of the priority-level=1 is assigned to all the tasks of this group. On other hand, the value of the priority-level = 2 is assigned to the second group that is inserted during the Monitor-Task operations.

In order to avoid starving for the tasks in the soft queue, the re-sorting of tasks according to the waiting times are performed only for the tasks of the priority-level=2. The re-sorting is not performed for the tasks of the priority-level=1. Therefore, when all the tasks of priority-level=1 are allocated to the VMs, the priority-level=2 is changed to priority-level=1 for all the tasks. But, the value of the priority -level =2 is assigned to the upcoming tasks.

The long tasks cluster includes only one type of tasks that is defined as Long-Tasks. Hence, for the free VMs, the tasks that have the smallest waiting times are allocated.

On the other hand, the short tasks cluster, includes two types of tasks. At first, the Short-Tasks are sorted according to their waiting times. Next, the Real-Tasks are sorted according to their deadline times. Hence, the VMs management in this cluster is performed according to the deadline of the real-time tasks and the smallest waiting time of the Short-Tasks. Based on the values of deadline and the smallest waiting time, three cases may be occurred for executing tasks in the short tasks cluster.

Based on the values of deadline and the smallest waiting time, three cases may be occurred for executing tasks in the short tasks cluster.

- When there are no Real-Tasks, the Short-Tasks are behaving as the tasks of the long tasks cluster. For the first free VM, the task that has the smallest waiting time is allocated.

- When there is a Real-Task and a Short-Task competing on the expected available VM.

- If the deadline of the Real-Tasks allows the Short-Tasks to be executed partly or completely, the Short-Tasks are submitted to the available VM. Otherwise, the Real-Task is submitted to the available VM.
- When the Deadline of the Real-Task will be spent without the available VM.
    - In this case, the Short-Task on the VM which has the shortest remaining time is suspended. Its status is buffered in the Short-Task buffer
    - Allocate the Real-Time task to the free VM
    - When the Real-Time task has been executed, the buffered Short-Task is resumed.

The procedures of the VM-manager have been clarified the following two main functions:

VM-Manager initiation

Step 1: Create two empty clusters

Step 2: Divide the amount of the share between the two clusters.

Step 3: Distribute the homogenous VMS to each cluster such that each cluster will have the half of the processing power of all the available VMs.

VM-Manager share-update

Step 1: Receive the notification of the share change from the Share-Ratio function

Step 2: Re-allocate the VMS between the two clusters according to the new share value

Step 3: The assigned tasks are to be the migrated VMs according to the old share must finish their task before implementing the migrations (i.e. the migration is not performed for the VM has a task).

Real-task-allocation

Step 1: If the new real task arrived, choose a VM which has the shortest remaining time $VM_{t.rem}$.

Step 2: If $VM_{t.rem}$ plus the expected execution time for the real task $t_r.exeTime$ will not break its deadline, let soft task to be finished.

$$t_r.deadline > VM_{t.rem} + t_r.exeTime$$

Step 3: If $VM_{t.rem}$ plus the expected execution time for the real tasks $t_r.exeTime$ greater than the deadline:

- Suspend the soft task and buffer it.
- Hold its status in the Short-Task buffer
- Assign the Real-time task to the free VM

Step 4: After the execution of the Short-Task the VM-Manager resume the execution of the holding Short-Task backing its status from the Short-Task buffer

IV. Performance Evaluation

In order to evaluate the experimental results, the WorkflowSim [20] is used to simulate the different methods of the scheduling. The WorkflowSim is an open source workflow simulator that extended the CloudSim [21]. The simulation assessment is performed by using the homogeneous characteristics in the fogs computations. They are based on the characteristics of the VMs in the Amazon EC2. So, each task is performed on a T2.Micro instance of Amazon EC2 that is available for free.

The proposed model was implemented in order to compare another four models. Firstly, the FCFS which is used to serve the tasks according to their arrangements that are based on the arrival time. In addition, another three compared models are already published for the cloud environment. They are the Max-Min, the PBATS and the RETS. The Max-Min maintains a task status table to anticipate the actual loads of the virtual machines and the estimated completion time of tasks, which can assign the workload among nodes [22]. The Priority Based Autonomic Task Scheduling (PBATS) that schedule its tasks according to three different priorities levels [23] [24]. On the other hand, the Real Time Efficient Scheduling (RETS) is based on allocating a ten percentage for the real-tasks. All these scheduling methods are compared by the proposed method to measure the load balancing in the proposed model.

The assessment has been performed in a two cases. The first case is implemented to measure the performance of the system for the soft tasks. The second case is to measure the reliability of the system for the real-time tasks. In both cases, the performance is measured based on three parameters. They are the average turnaround time, the average waiting time and the throughput. These parameters are measured with the existing tasks for the proposed model and all the other compared scheduling methods. In addition, all the following comparisons are performed using 10 VMs of the Amazon EC2. The VMs are homogenous of processing power of 2000 MIPS while the sizes of the executed tasks are different from 2000 to 6000 instructions.

The following subsections are organized into two subsections. The first subsection measures the performance of the system using all types of tasks. The second section measures the effect of the system on the real time tasks only.

A. *performance measurment for All types of tasks*

In the next subsection, the effect of the number of all tasks on the response time is tested. Also, subsection II measures the waiting time of the system. Finally, subsection III measures the throughput.

1. *Turnaround Time performance Test*

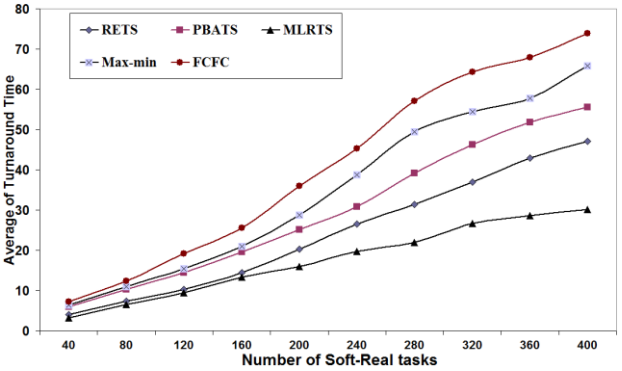

Fig.4. Turnaround Time Soft-Real Tasks Test.

The first experiment is performed based on the Turnaround time parameter. The proposed model (MLRTS) is compared to the whole mentioned four algorithms. All of the experiments are performed using 10 different workloads from 40 to 400 tasks. The real time tasks will represent 20 % from all of the inserted workload in each experiment. The task sizes are ranged from 2000, 8000. The sizes of the tasks less than 3000 are considered as the Short-Tasks. The tasks of the sizes more than 3000 are considered as the Long-Tasks. All of the tasks are executed based on the ten VMs of processing power of the 2000 MIPS for each machine. The obtained results that represent the average turnaround times of all of the algorithms are shown in Figure 4.

It is clarified that, the curve that represent the average of the turnaround time of the FCFS is rapidly increased. Especially, as the number of the tasks are increased. This increase is caused due to the non-preemptive property of that algorithm. The non-preemptive property of the long tasks at the front of the queue causes the short tasks at the end of the queue to wait for more time. Also, the Max-Min curve is the nearest curve to the FCFS curve. Actually, the Max-Min algorithm is based on the assigning the longest tasks to the VMs that have minimum remaining execution time. So, the short tasks will be waited for a long time to get the VMs to be executed. The delays accumulation of the short tasks will affect the average amount of the turnaround time for the Max-Min curve. In addition, the PBATS curve is revealing a less in the average turnaround time results as compared to the FCFS and Max-Min. Actually, the tasks of the PBATS algorithm are exposed to three levels of

priorities. Hence, the different sizes of the performed tasks are affected by these priority levels.

On other hand, the curve of the RETS refer to reasonable results with a light load up to 150 of tasks. However, as the number of the tasks increase, an inefficient performance is occurred if compared by the proposed algorithm (MLRTS). The RETS ineffective is caused due to the 10% of the resources assigned to the real tasks. This constant percentage presents a problem if there are no adequate real tasks. Also, it is a problem if the real tasks exceed the assigned resources. Moreover, there is no priority or classification strategy in the RETS.

Actually, the MLRTS doesn't suffer from these problems. It is designed based on the permanent balance between the resources of the fog center and the expected workload within the fog center. In addition, the overload tasks are instantly directed to one of the neighboring fogs or to the related cloud. I.e. this ratio doesn't measure in the average of the turnaround time computation.

Moreover, the MLRTS efficiency is not affected by the excess in the workload. Thus, the MLRTS doesn't upset the scalability of the system and offers the less amount of the turnaround time. Accordingly, the MLRTS is the most efficient algorithm among all of the Scheduling Algorithms in the fog and cloud environment.

2. The Waiting Time Performance Test

In this experiment, the performance is measured based on the average of the waiting time parameter. The experiment is performed based on the same work load of the previous experiment. The obtained results for all the algorithms are shown in Figure 5. It is revealed that, the average waiting time of the FCFS is rapidly increased. Especially, as the number of the tasks are increased. The reasons of the increasing are similar to the reasons that cause the increase in the average turnaround times. Also, the curve represents the Max-Min algorithm is the nearest one to the FCFS curve. The results of this curve are logically accepted due to the increase in the waiting time of the short tasks. The accumulation waiting time of these tasks will increase the total average waiting time that is computed for all the tasks. For the PBATS, the tasks are distributed according to the three levels of priorities causing the rations from the incoming tasks to wait for long time according to their levels. The increase in the waiting times of these tasks causes the increase in the sum of the average waiting time.

On other hand, the RETS algorithm gives reasonable sum for the average waiting time for the workload less than or equal to 100 tasks. But, when the work load increases, the average waiting time is rapidly increased. This increase is reasoned due to the unbalanced state between the amount of the real time tasks and the quantity of the allocated resources. The restricted quantity of the resources make a lot of the real time tasks lose their deadline times. This problem has been overcome by the the MLRTS as shown from the results applied on its curve. The MLRTS is employeed to serve the real-time tasks and the short-tasks on the same VMS and the long-tasks are served on the other VMs. This division guarantees no idle VMs while the tasks of both of the types waits for the execution. Moreover, the Share-ratio is re-computed according to the change in the load types to accomplish the required balancing between the tasks and the resources. So, the shortage that may occur in the resources of the soft tasks are balanced by the resources of neighboring fogs computing centers or the related cloud.

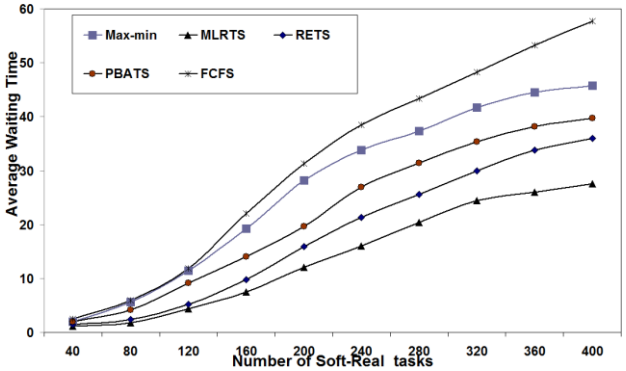

Fig. 5 Waiting Time Soft-Real-Task Test.

3. The Throughput Performance Test

In this experiment, the performance is measured based on the average of the throughput parameter. The throughput is computed as the number of the executed tasks per time. Also, the experiment is performed based on the same workload of the previous experiments. The obtained results for all of the algorithms are shown in figure 6. It is shown that, the throughput of MLRTS is clearly increased than the throughputs of the other algorithms. In addition, the throughputs of the FCFS are the nearest curve to the MLRTS. The combination of short and long tasks of the FCFS will increase the number of the executed tasks. In addition, the Max-Min and PBATS are fallen back as the lower throughputs. Both of these algorithms will execute the tasks of the long sizes at first. So, the number of the completed tasks will be reduced than other algorithms. Hence, the Max-Min and PBATS throughputs are minimized due to the computation that based on the number of output tasks/time. In addition, the RETS throughput is fallen in an intermediate range among all the throughputs for all the compared algorithms. The RETS results are expected due to the unbalance between the loads types and the distributed resources.

Therefore, the powerful advantage of the MLRTS is occurred due to the flexibility in allocating the Short-Tasks to a specific cluster. This allocation will lead to a huge number of the complete execution for the Short-Tasks. Especially, if there is no real tasks or a small number of them. In addition, the MLRTS has a great flexibility in re-assigning the available resources according to the needs of the load type which increase the throughput without effect on the fairness of the load balancing.

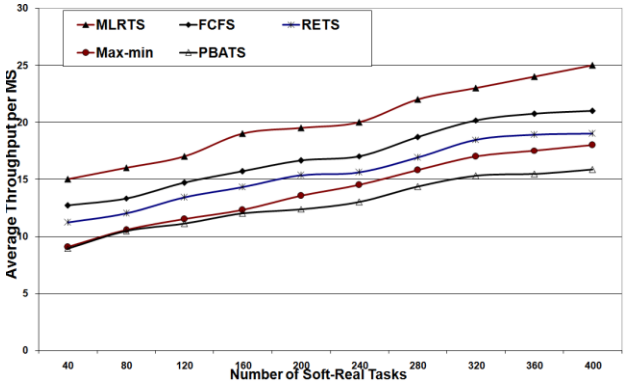

Fig. 6 Average Throughput of Soft-Real-Task Test.

B. *Real-task performance test*

This test is implemented to measure the effect of the proposed system on the real time tasks. The experiments are repeated for 10 times. Each experiment is performed on the percentages of 20% to 25% from the inserted workload. Through the experiments, the workloads for all the tasks types are changed from 40 to 400 tasks. According to the used percentage, the real time tasks are changed from 10 to 80 tasks. The sizes of the performed tasks are alternated according to the ranges that mentioned before. However, all the experiments of the Real-Time tasks are performed in the presence of the soft tasks load.

The following subsections will measure the effect of proposed model on the Real-time tasks compared with other algorithms. The comparisons were performed based on the previous measured parameters. Subsection I measures the turnaround time. Subsection II measures the waiting time and Subsection III measures the throughput. The final section will reveal some snapshots for the states of the Share-Ratio computations.

1. *Real-task Turnaround test*

The results obtained from the experiments that run for the average turn aroundtimes of the compared algorithms are shown in Figure 7. The worst results are shown by the curves that represent the FCFS, PBATS and the Max-Min respectively. However, the main disadvantage of the previous algorithms is the incapability to serve the real time tasks according to their deadline . I.e. the deadline times are not considered when the tasks are arranged for execution. Therefore, the real times tasks are handled as the soft tasks handling In addition, the RETS introduces a good behavior when the number of the Real-Tasks are suited to the 10 percentage of the resources that are allocated for the real-time tasks. In the figure, the good result is obtained for the number of 40 tasks. But, bad results are obtained if the amount of the real-tasks is increased. On the other hand, in MLRTS, it is not allowed for the real time tasks to wait more than their deadline times. In addition, it is not allowed for the real time tasks to migrate to the neighbor fogs to get their services. So, the real time tasks are not exposed to any delay which minimizes the total average of turnaround time. So, the less turnaround time is obtained by the MLRTS. In addition, it is the most concerned system that reduce the turnaround time when the number of the Real-time tasks is increased

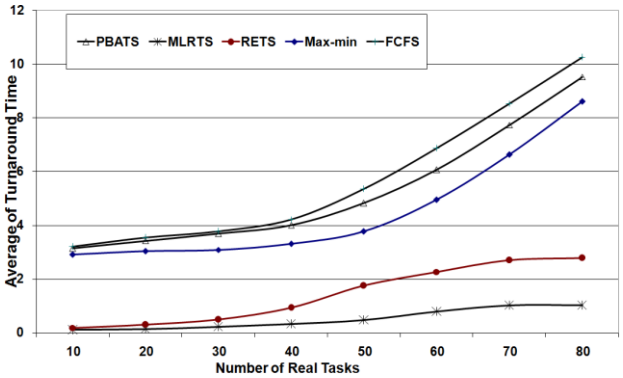

Fig.7. Real-Tasks Turnaround Time Test

2. *Waiting Time Test for Real Tasks*

The averages of waiting time curves that reveal the effect of the proposed model on the execution of the real tasks are shown in figure 8. In this figure, the least average waiting time is revealed for MLRTS. Actually, the proposed model is designed to introduce the highest priority for the Real-Time tasks. So, the resources may be migrated from the Long-Task cluster to accomplish the sufficient resources that may be needed by the Real-Times tasks. However, the RETS curve is the nearest one among all the compared algorithms to the MLRTS. But, the increase in the number of real tasks will cause the overhead of the intermediate layer in RETS which increase its average waiting time. Also, using the traditional scheduling methods that does not consider the deadline times of real tasks such as FCFS, Max-Min and PBTES will increase the average waiting time. Hence, the deadline times of the real tasks may be broken.

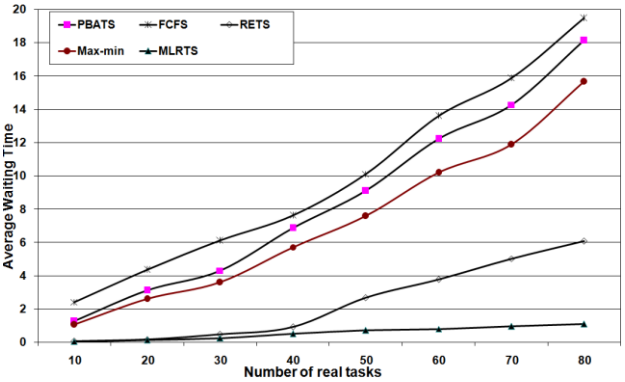

Fig.8. Waiting Time Test for Real Tasks

3. *Throughput Test for Real tasks*

The resultant curves of the throughputs for the compared algorithms are shown in figure 9. It is clear that, the high amount of the throughput is achieved by the MLRTS. The throughput results fluctuates between 97% and 100% for all the inputs of the different loads for the real tasks. MLRTS combines the Short-tasks and the real-

time tasks on the same cluster. This combination offers the guarantee for the Real-Time tasks to obtain their needed resources. The real tasks gains the highest priority more than the Short-tasks. Moreover, the execution of the Short-tasks should be suspended to inhibit the deadline time of the real tasks from the broken. So, the throughput of the real time tasks will be increased for MLRTS.

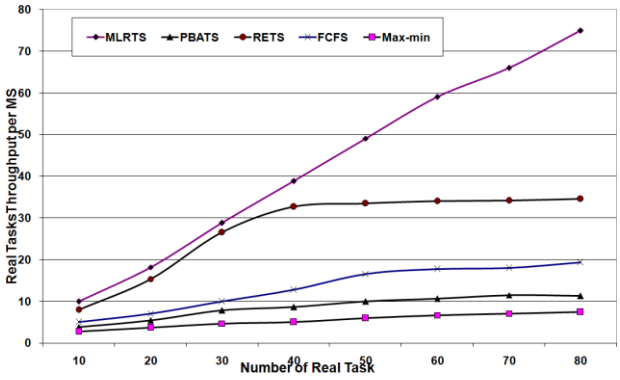

Fig.9. Real Throughput

On other hand, the throughput curve of the RETS gives accepted results when the input include a small number of the real tasks. But, when the input number of the real tasks increases, the results are reduced to less than 50% from the input Real-Tasks. The reduction of the RETS throughput is caused due to two reasons. The first reason is the constant percentage (10%) of the resources that is reserved to serve the real tasks, and the second reason is the migrations of the excess real tasks to the neighboring fog to get its service. These migrations will consume the time allowed to the deadline time before the service is accomplished. However, the throughputs of the Max-Min, PBATS and the FCFS algorithms are minimal. These results are logically accepted. The last three algorithms are not designed to serve the real tasks. So, each real task served in these algorithms is performed based on criteria doesn't deal with deadline times. We can say, it is served by chance.

C. Share Resources Re-Distribution

In this section, some of the successive snapshots for the test simulation of the Share-Ratio re-computation are introduced. These snapshots give a set of the re-computations for the Share–Ratio based on the change in the number of the different tasks types according to the σ_L and σ_{SR}.

According to the table 1, the system starts at the stable case (state 1). At state 2, the maximum waiting time is increased for the Long tasks to 1.2 second. So the L-flag will be changed to 1 and the incoming long tasks are converted to the FCB.

Accordingly, the resources will be re-distributed again between the long-task cluster and the short-real-task cluster. Hence, the number of VM in the long cluster has been increased on account the Short-real-task cluster. After changing σ_L at state 4, the load of the Real-Short-tasks is increased and long-tasks are decreased. So, the system will adjust itself by re-computing the share-ratio based on the new σ_L and σ_{SR}.

Table 1. Share Resources Re-Distribution

state	Num Long tasks	Num Real Short Tasks	σ_L	σ_{SR}	L-Flag	S-Flag	VMs L-cluster	VMs S-cluster
State 1	6	15	1 S	1 S	0	0	5	5
State 2	12	8	1.2 S	1 S	1	0	6	4
State 3	4	5	1.2 S	1 S	0	0	6	4
State 4	3	21	1 S	2 S	0	1	3	7
State 5	4	14	1 S	2 S	0	0	3	7
State 6	9	17	1 .3 S	2.1 S	1	1	3	7
State 7	11	22	1.2 S	2 S	1	1	3	7
State 8	4	10	1 S	1.5 S	0	0	4	6

However, at some circumstances the load increases suddenly and both of L-flag and S-flag are changed to 1 value as in state 6. In this case, the incoming tasks from all the types of the soft tasks are directed to the FCB. The Real-Tasks are the only received tasks. All the tasks within the soft queues are assigned to priority-level 1. The values of the L-flag and S-flag are kept to the 1 value until the value of the σ_L or σ_{SR} becomes less or equal to its previous value due to the execution of the tasks that are currently in their queues. At this moment the Share-Ratio is re-computed again and VMS are re-distributed accordingly.

V. Conclusion And Future Work

In this paper, the MLRTS model is proposed as multi-level scheduling that accomplishes the load balancing in the data fog center and their neighboring fog and related cloud. The model was designed for efficient reserving of the fog resources under different types of soft and real tasks. The proposed model offers an elastic re-allocation for the available processing power of the VMs according to the change in the load of the different types of the tasks. In addition, it provides an adaptable allocation for real time tasks due to their dead line times. The soft tasks are sorted in their queues based on the SJF scheduling. Therefore, the MLRTS model is implemented based on the two priority-levels for each soft type of the tasks. The priority-levels are used to inhibit the starving of the tasks within their queue type. The proposed model includes an independent module defined as FCB. This module is implemented as an interface to manage the transfer of the tasks that will not serve in its local fog to one of the neighboring fogs or the related cloud to be served. In the future, we have two targets. The first one is offering a re-engineer for the system to be more suitable for live video. The second target is the development of this model to benefit from the facilities of the heterogeneous processing power.

REFERENCES

[1] Chandrasekhar S. Pawar, Rajnikant B. Wagh,‖ Priority Based Dynamic resource allocation in Cloud Computing with modified Waiting Queue‖, *Proceeding of the IEEE 2013 International Conference on Intelligent System and Signal Processing(ISSP)* Pages 311-316.

[2] Yusen Li, Xueyan Tang, Wentong Cai,‖ Dynamic Bin packing for on demand cloud resource allocation ‖, *Proceedings of the IEEE Transactions on Parallel and Distributed Systems* ,2015,Paged 1-14.

[3] Kamyab Khajehei, ―Role of virtualization in cloud computing‖, *International Journal of Advance Research in Computer Science and Management Studies* Volume 2, Issue 4, April 2014.

[4] Savani Nirav M, Prof. Amar Buchade, ―Priority Based Allocation in Cloud Computing‖, *International Journal of Engineering Research & Technology (IJERT)* ISSN: 2278-0181 IJERTV3IS051140 Vol. 3 Issue 5, May – 2014.

[5] Brototi Mondala, Kousik Dasguptaa, Paramartha Duttab"Load Balancing in Cloud Computing using Stochastic Hill Climbing-A Soft Computing Approach", Elsevier, Procedia Technology 4(2012) pp. 783 – 789.

[6] Ivan Stojmenovic, sheng Wen, "The Fog Computing Paradigm: Scenarios and security issues" Proceedings of the IEEE International Fedrerated Conference on Computer Science and Information Systems, 2014, pp.1-8.

[7] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, Sateesh Addepalli "Fog Computing and its Role in the internet of things",http://conferences.sigcomm.org/sigcomm/2012/paper/mcc/p13.pdf.

[8] Manisha Verma, Neelam Bhardwaj Arun Kumar Yadav," An architecture for load balancing techniques for Fog computing environment", International Journal of Computer Science and Communication, Vol. 8 • Number 2 Jan - Jun 2015 pp. 43-49.

[9] S. F. El-Zoghdy and S. Ghoniemy, "A Survey of Load Balancing In High-Performance Distributed Computing Systems", International Journal of Advanced Computing Research, Volume 1, 2014.

[10] Mohsen and Hossein Delda, "Balancing Load in a Computational Grid Applying Adaptive, Intelligent Colonies of Ants", Informatica 32 (2008) 327–335.

[11] Antony Thomas, Krishnalal G, Jagathy Raj V P,"Credit Based Scheduling Algorithm in Cloud Computing Environment", International Conference on Information and Communication Technologies, Procedia Computer Science 46(2014) 913-920.

[12] Dhinesh Babu L.D, P. Venkata Krishna,"Honey bee behavior inspired load balancing", Elsevier, Applied Soft Computing 13(2013) 2292-2303.

[13] Brototi Mondala, Kousik Dasguptaa, Paramartha Duttab"Load Balancing in Cloud Computing using Stochastic Hill Climbing-A Soft Computing Approach", Elsevier, Procedia Technology 4(2012) pp. 783 – 789.

[14] Atul Vikas Luthra and Dharmendra Kumar Yadav,"Multi-Objective Tasks Scheduling Algorithm for Cloud Computing Throughput Optimization", International Conference on Intelligent, Communication & Convergence, Procedia Computer Science 48(2015) 107-113.

[15] Manisha Verma, Neelam Bhardwaj, and Arun Kumar Yadav, "Real Time Efficient Scheduling Algorithm for Load Balancing in Fog Computing Environment", International Journal of Information Technology and Computer Science, Vol.4, No.2, pp.1-10, 2016. DOI: 10.5815/ijitcs.2016.04.01

[16] Po-Huei Liang and Jiann-Min Yang,"Evaluation of two level global load balancing framework in Cloud Environment", International Journal of Computer Science and Information Technology (IJCSIT), Vol. 7 No 2, April 2015.

[17] Mohamed A. Elsharkawey, Hosam E. Refaat,"CVSHR: Enchantment Cloud-based Video Streaming using the Heterogeneous Resource Allocation", International Journal of Computer Network and Information Security (IJCNIS), Vol.9, No.9, pp.1-11, 2017.DOI: 10.5815/ijcnis.2017.09.01

[18] M.Verma, N. Bhardwaj and A. Kumar, "Real Time Efficient Scheduling Algorithm for Load Balancing in Fog Computing Environment",I.J. Information Technology and Computer Science, April, 2016, 4, 1-10

[19] Himani and Kamaljit Kaur," Deadline Scheduling in Cloud Computing: A Review", International Journal of Computer Applications(0975-8887),Vol. 96-No.24,\june 2014.

[20] W. Chen and E. Deelman, ―Workflowsim: A toolkit for simulating scientific workflows in distributed environments, in 2012 IEEE 8th International Conference on E-Science, ser. eScience, 2012, pp. 1–8. [Online]. Available:https://github.com/WorkflowSim

[21] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, ―CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience, vol. 41, no. 1, 2011.

[22] Xiaofang Li, Yingchi Mao, Xianjian Xiao, "An improved Max-Min task-scheduling algorithm for elastic cloud", Computer, Consumer and Control (IS3C), 2014 International Symposium on

[23] B.Anju and C.Inderveer (2016), "Multilevel Priority-Based Task Scheduling Algorithm for Workflows in Cloud Computing Environment". In Proceedings of International Conference on ICT for Sustainable Development: Volume

[24] Swati Agarwal, Shashank Yadav, Arun Kumar Yadav,"An Efficient Architecture and Algorithm for Resource Provisioning in Fog Computing", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.8, No.1, pp.48-61, 2016. DOI: 10.5815/ijieeb.2016.01.06

Authors' Profiles

Mohammed A. El-Shrkawey received his B.Sc. in Electrical engineering from the Military Technical Collage, Cairo in 1987. He received his M. Sc. in Computer Engineering from Faculty of Engineering, Al Azhar University, Cairo in 2002. He received his Ph. D. in Network Security from Faculty of Computers& Informatics, Cairo University in June 2007. He is currently a lecturer in Faculty of Computers & Informatics, Suez Canal University. Ismailia, Egypt. His current research interests are Networks, Modeling, simulation, and Image Processing

Hosam E Refaat: has graduated from the Faculty of Science, Assuit University, Egypt, in 1998. In October 2006, he finished his Master degree in the field of distributed systems from the faculty of Science, Cairo University, Egypt. Currently, he is a lecturer in Faculty of Computers & Informatics, Suez Canal University. Ismailia, Egypt. His current research interests are Parallel Systems, Cloud Computing, Edge Computing, and Data mining.

How to cite this paper: Mohamed A. Elsharkawey, Hosam E. Refaat, " MLRTS: Multi-Level Real-Time Scheduling Algorithm for Load Balancing in Fog Computing Environment", International Journal of Modern Education and Computer Science(IJMECS), Vol.10, No.2, pp. 1-15, 2018.DOI: 10.5815/ijmecs.2018.02.01

