DLBS: Decentralize Load-Balance Scheduling Algorithm for Real-Time IoT Services in Mist Computing

Hosam E. Refaat[1]
Dept. of Information System
Faculty of Computers and Informatics
Suez Canal University, Egypt

Mohamed A.Mead[2]
Dept. of Computer Science
Faculty of Computers and Informatics
Suez Canal University, Egypt

***Abstract*—Internet of Things (IoT) has been industrially investigated as Platforms as a Services (PaaS). The naive design of these types of services is to join the classic centralized Cloud computing infrastructure with IoT services. This joining is also called CoT (Cloud of Things). In spite of the increasing resource utilization of cloud computing, but it faces different challenges such as high latency, network failure, resource limitations, fault tolerance and security etc. In order to address these challenges, fog computing is used. Fog computing is an extension of the cloud system, which provides closer resources to IoT devices. It is worth mentioning that the scheduling mechanisms of IoT services work as a pivotal function in resource allocation for the cloud, or fog computing. The scheduling methods guarantee the high availability and maximize utilization of the system resources. Most of the previous scheduling methods are based on centralized scheduling node, which represents a bottleneck for the system. In this paper, we propose a new scheduling model for manage real time and soft service requests in Fog systems, which is called Decentralize Load-Balance Scheduling (DLBS). The proposed model provides decentralized load balancing control algorithm. This model distributes the load based on the type of the service requests and the load status of each fog node. Moreover, this model spreads the load between system nodes like wind flow, it migrates the tasks from the high load node to the closest low load node. Hence the load is expanded overall the system dynamically. Finally, The DLBS is simulated and evaluated on truthful fog environment.**

***Keywords*—Cloud computing; fog computing; mist computing; IoT; load balancing; reliability**

I. Introduction

Cloud computing is presented as an ongoing innovation, which is totally dependent on the web. The engineering of the Cloud computing depends on a focal server that keep up a tremendous measure of sharing database, various assets and an enormous number of business applications. Then again, a colossal number of remote customers that has a place with various associations can profit by the various administrations given by the focal server. Every remote client has its own, working framework and internet browser that work autonomously on the substance of the cloud server [1, 2]. The association of the client to the web is the main prerequisite from the client to use the cloud server capacities. Along these lines, the IT business and any little association can get these services from the cloud without spending tremendous measure of cash in equipment or software. As a matter of fact, the execution of the cloud introduces a few related ideas. These ideas manage virtualization, resource allocation, computing distribution, utilization of bandwidth, load balancing, fault tolerance, high availability and dynamic scalability for various classifications of data and applications. The administration of the operations identified with every one of these concepts is performed by the cloud service provider.

The cloud providers allocate the resources to the end clients as a service relying upon the uniqueness of the service models and furthermore dependent on the client needs. The service models may incorporate Software as a service known as SAAS, Platform as a service known as PAAS, Infrastructure as a service known as IAAS. These services are inclined on one another and in a pool way.

By and large, the executions of the various procedures on the cloud present a few advantages to the end clients. At First, the data is shared more than one stage, so better services are conveyed to every user. Also, the end user can get the services resources on-demand, flexible, reliable and portable way as indicated by his need as it were.

In spite of these advantages that can be offered by cloud computing to enormous applications, it faces a lot of challenges [3]. The first challenge happens when the number of the clients is increased. For this situation, the requests are broadened to increase the number of services than the cloud capacities. As client requests is increased, as the responses time is increased unless the available resources and the available bandwidth are upraised to acquire all the extra requests. The second challenge happens when the created data by the cloud services is migrated through a long distance from the cloud to the clients. The far distance creates additional challenge about the data security. Moreover, an unpredictable abundance in the workload may cause the need to create a novel load balancing strategy. The load balancing is the reasonable assignment of the task among the parallel resources such as networking, hard drives and computers [4]. In this way, it will be required to achieve the improvement in the distribution of the computation resources and storage devices. So as to beat these challenges, another innovation of profoundly virtualized processing model has been displayed known as Fog computing. The model [5] is proposed by CISCO to be held as cloud edge of an enterprise organizes. The control of the fog computing isn't a substitution of the cloud computing. In reality, it fills in as a steady domain that can give high QoS to the diverse client requests of the close distances. In this way, the entire fog-cloud colony comprises of a set of fogs computing servers and a set of the clouds computing centers.

For the most part, the activities of the Fog processing are like the cloud computing with two fundamental differences. The principal difference is identified with the area of the fog computing that is put near the clients. Subsequently, the fog computing can be envisioned as a nearby cloud. The second difference related to the resources capacities of the fog that have fewer capacities contrasted with the capacities of the cloud assets. In any case, each fog computing incorporates its very own server that is bolstered by its own resources. Furthermore, each fog server is involved by the vital software or firmware to set up the required VMs, for example, the hypervisor. Whilst Cloud computing exhibits big data processing at the data center level, fog computing provides data processing and actuation capability at the network edges [6, 7]. Also, Fog computing expand the same capability in the middle at edge gateways. In another word, fog computing provides the closed resources to many services, which cannot be realized with alternative strategies [8, 9].

The scattered IoT devices create the need to spread the fog nodes to cover the IoT environment. As of late, mist computing has been rise to capture a more extreme edge [10]. In other words, the nodes in the fog environment are classified as mist and middle edge node, as shown in Fig. 1. The mist computing model depicts scattered computing at the extreme edge. It has proposed with future self-aware and autonomic systems in mind [11]. The Mist server can exist with the Internet Service Provider (ISP) or separately in the network. Mist computing is proposed as the first computing node in the IoT-fog-cloud colony; it can be called as “IoT computing” or “things computing”. An IoT device may be portable like smart watch, a mobile device, or stationary like a smart AC.

Generally, the Load balancing seems to assume an imperative for scheduling the various types of the users’ tasks. Load balancing can be characterized [12] into different categories such as the applied state that maybe static or dynamic, the load balancer techniques which is hardware or software and the policies rules such as resource, information, selection, location and transfer. The workload in the static load balancing approach is based on the current performance of the processing nodes with careless about future changes. Moreover, in this approach the waiting tasks can't migrate from its processing nodes [13]. Also, the static load balancing methods treat the tasks in non-preemptive manner. Otherwise, the dynamic load balancing decides the tasks distribution during the run time based on the information of system status [14]. In this way, the task scheduling algorithm is employed to reserve the resources to the IoT devices on servers to satisfy the fair distribution. The satisfaction of the fairness will reduce the task waiting time. Furthermore, it will enhance the tasks execution speed using the free resources and optimum consumption of storage to minimize the turnaround time of the submitted tasks.

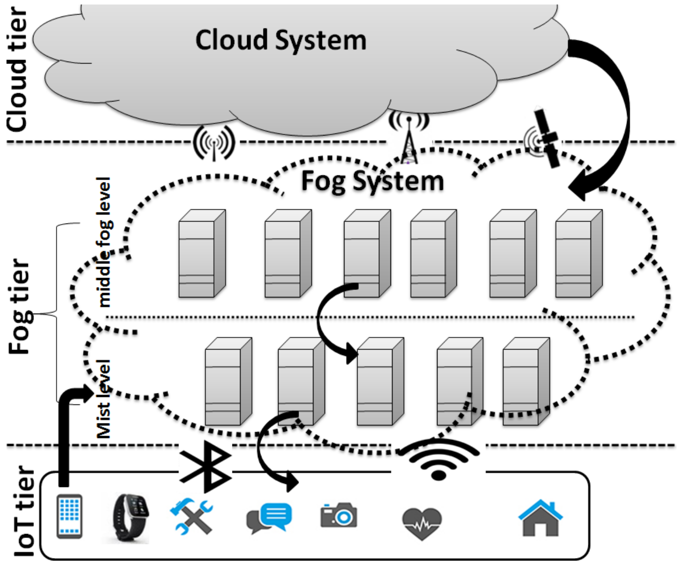

Fig. 1. IoT with Edge Computing and Cloud.

The proposed model in this paper is based on a dynamic load balancing algorithm. This model gives the real tasks the first priority to fit its deadline. Also, the real tasks can migrate from mist node to the others to avoid missing the deadline time. On other hand, the system preserves a specific quality of service (QoS) for each type of soft task requests. Moreover, the proposed load balancing algorithm is acting as a wind flow. It migrates the load of service requests from the high load nodes to the low load nodes. This strategy minimizes the communication overhead in spite of the user task migration. Hence, each node cooperates with the others nodes to maintain balanced load among them.

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 performance evaluation and the results of the simulations are introduced in sections IV and V. Section VI conclude the paper and provide the venues for the future work.

II. Related Work

In this segment, Several Load Balancing algorithms are presented for different authors. These algorithms are investigated dependent on the diverse parameters, for example, due date, execution time, data transmission, cost, need, dependability, adaptability, task length and throughput. Basically, the effective load balancing algorithms have been implemented in the cloud system.

Generally, the load balancing mechanisms in both of the cloud and Fog is same with just principle distinction. In the fog computing, the load balancing should maintain system more feasible and effective with in spite of resources limitation. It offers access to the assets of less transmission capacity and time. In this way, the mist computing has fulfilled the requirements for the closest IoT at a gigantic rate with no disarray like what may happen for the network traffic.

In this area, the first load balancing technique is introduced in [15]. This method is intended to achieve good services by increasing the resource utilization based on two parameters, which are the task priority and its length. The choice of the tasks for the scheduling might be gotten from both of the first and last indexed queue to accomplish an all the more relentless framework.

The tasks are scheduled dependent 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 processing is given to the high credit task. However, this algorithm suffers from certain shortcomings when the absolute credits of several tasks became indistinguishable. For this situation, the FCFS has to be added without guarantee of tasks to be completed earlier or to its deadline.

Another algorithm depends on comparable to conduct of honey bee model (HBB-LB) is proposed by Dhinesh babu L.D et al. [16]. In this algorithm, the priority is taken as a fundamental QoS factor to Bar any procedure from hanging tight for quite a while in the line to diminish the execution time and augment the throughput. Similarly, the tasks can be acted as the Honey bees and the Virtual Machines can be acted as sustenance sources. Moreover, The VMs are classified according to three circumstances, balanced overload, high overload and low overload. When the VMs are overloaded, the tasks are evacuated and act as a honey bee. So, these tasks are migrated to the low load VMs. These duties are depending on how many high priority tasks are executed on those VMs. It should be noticed that the VM is chosen based on the low overload and the least number of the executed priority tasks. After proper tasks on VM, data is refreshed with the goal that the rest of the assignments can acquire their needs under load VM. This algorithm has presented certain advantages represented in the proper resource utilization; maximizing the throughput while keeping different QOS parameters which are built on the task priority. On the other hand, the disadvantages are introduced for the low need priority tasks which suffer from idle state or long time waiting in the queue. These tasks may be dismissed causing the unbalancing of the workload balancing.

For an enormous scale condition, e.g., cloud computing framework, there had been also various scheduling approaches proposed with the objective of accomplishing the better task execution time for cloud resources [17]. Independent task scheduling algorithms mainly include MCT algorithm [18], MET algorithm [15], MIN-MIN algorithm [15], MAX-MIN algorithm [19], PMM algorithm, and genetic algorithm. The MCT (Minimum Completion Time) algorithm assigns each task in any order to the processor core that causes the task to be completed at the earliest time. It prohibits some tasks to be allocated to the fastest processor core. The MET (Minimum Execution Time) algorithm allocates each task to a processor core in any order that minify the task execution time. As opposed to the MCT algorithm, the MET algorithm does not consider the processor core's ready time, which may prompt genuine burden unevenness crosswise over processor cores. The MIN-MIN algorithms calculates the minimum completion time of all unscheduled tasks firstly, and then chooses the task with the minimum turnaround time and allocate the task to the processor core that can minimize its turnaround time, repeating the process many times until all tasks are allocated. The same as the MCT algorithm, the MIN-MIN algorithm is also based on the minimum completion time. The MIN-MIN algorithm proposes all tasks that are not scheduled, but the MCT algorithm considers unique task at a time. The MAX-MIN algorithm is similar to the MIN-MIN algorithm, which also computes minimum completion time without scheduled tasks firstly and then selects the task with the largest minimum completion time and assigns the task to the processor core with the minimum completion time.

Mondala et at. use an optimized approach algorithm to have load balancing scheduling system [20]. This model is based on a centralized load balancing algorithm. In another words, the system is based on a central node that distributes the workload tasks. Hence, the main drawback is of this model is that if the central node fails, the whole working of the system will fail. This means that the central node is represent the system bottleneck. So here, using decentralized load balancing strategy solves this bottleneck. Resource utilization can be done effectively to enhance the throughput, accordingly decreasing the cost of an application running in a SAAS environment without break service level agreements [21].

Actually, the different scheduling algorithms based on QoS parameters have been introduced for different environments in [22]. The scheduling is performed to achieve the huge service requests and to enhance the efficiency of the workload. Subsequently, there are numerous modules that are implemented in each kind of the scheduling algorithms, for example, Min-Min, FCFS, Max-Min, Round-Robin algorithm.

Nevertheless, the one of the efficient methods among them is the heuristic method. Its allocating the tasks includes three stages in a cloud computing. At first, the VMs are located. Hence, the best target VM is chosen. At last, the task is assigned to the target VM. Lately, the Real Efficient Time Scheduling (RETS) is investigated in [23]. The main goal of RETS is to process the real-time tasks without delay. Therefore, it keeps one tenth of the available resources for the real-time tasks. Although, this ratio can be insufficient if the real-time tasks exceed this ratio. On the other hand, one tenth of the available resources will be idle if there are no real-time tasks.

Moreover, Anju et al. introduces multilevel of priority-based task scheduling algorithm (PBATS)[24, 25]. This algorithm has three levels of priorities, which prioritizes the tasks based on the length of the instructions. Also, to enhance performance of PBATS, it migrates the tasks under the minimum migration time policy. This policy can cause overload of node, which has low network overhead. Also, this policy doesn't distinguish between the real and soft tasks.

Also, Wang et al. proposed a task scheduling algorithm in the fog computing, which is called "hybrid heuristic (HH)" algorithm [26]. HH algorithm is mainly focus in solving high energy consumption in case of using limited computing resources. Unfortunately, HH method isn't distinguish between the mist and middle fog nods. Hence, this algorithm is not efficient method for real-time services.

III. PROPOSED MODEL

In a Fog computing environment, the load balancing is a pivot point for effective and efficient resource utilization, bandwidth and to achieves desired quality of service (QoS). Fog Computing system is divided virtually in two type of nodes, namely; mist and middle edge node. Actually, both types of fog nodes can have the same structure and resources. Nevertheless, the most closed node to IoT is called mist. Each Mist computing server is centered in the specific location mainly to receive the clients or/and IoT requests in a specific region. The fog colony is connected to a cloud system in the case of fog resources shortage to overcome the fulfillment of task requests.

In this paper, the new scheduling model (DLBS) is proposed in the cloud-fog-mist environment. The structure of this model is shown in Fig. 2. First of all, the Service Listener (SL) receives the user/IoT service request. Hence, SL creates a task for the service request and sends it to Load Balancing Allocator (LBA) module with required software from service container. Also, SL send task-metadata like, task type (real time or soft), expected execution time, etc. So, each Mist server is supplied by its own Load Balancing Allocator module (LBA). LBA is responsible for allocating the clients and/or IoT service requests into the fog resources. There two types of user/IoT request; real time and soft-tasks. The proposed model is designed to handle both types of tasks.

Mist node gives the real time task queue in resources allocation. The tasks in the real time queue will be allocated into one of idle local VMs in the node. If there is no idle, LBA preempt one of soft task VMs. In the worst case scenario, if there are no idle or soft VMs, ***Fog explorer*** module suggest the resources in the closest mist/middle edge node. Fog explorer detects the status of the other fog node by getting the status flags. The status flags are set by LBA module and broadcasted by the fog explorer. Each Mist node has four types of status flags, which determine the status of the node, namely, ***load lock***, ***real task lock***, ***receive status***, and ***send status***. Load lock flag, which is soft task waiting, is set by zero if the expected waiting time will not exceed QoS threshold (λ). In another word, λ grantees that the service of the soft tasks will be provided in a reasonable delay. If load lock flag is set by one, this fog node can't receive a soft task from other fog and its soft tasks will migrate outside the node. Also, real-time task lock is set by one if all VMs are allocated by real-time tasks. For any fog node if one of VMs is processing a soft task, the real-time task lock is set by zero. Finally, according to task migration the fog node blocks the receiving tasks from other nodes if its receive status or send status has value one. Obviously, the status flags are used to maintain the system balanced and available.

Mist node gives the real time task queue in resources allocation. The tasks in the real time queue will be allocated into one of idle local VMs in the node. If there is no idle, LBA preempt one of soft task VMs. In the worst case scenario, if there are no idle or soft VMs, ***Fog explorer*** module suggest the resources in the closest mist/middle edge node. Fog explorer detects the status of the other fog node by getting the status flags. The status flags are set by LBA module and broadcasted by the fog explorer. Each Mist node has four types of status flags, which determine the status of the node, namely, ***load lock***, ***real task lock***, ***receive status***, and ***send status***. Load lock flag, which is soft task waiting, is set by zero if the expected waiting time will not exceed QoS threshold (λ). In another word, λ grantees that the service of the soft tasks will be provided in a reasonable delay. If load lock flag is set by one, this fog node can't receive a soft task from other fog and its soft tasks will migrate outside the node. Also, real-time task lock is set by one if all VMs are allocated by real-time tasks. For any fog node if one of VMs is processing a soft task, the real-time task lock is set by zero. Finally, according to task migration the fog node blocks the receiving tasks from other nodes if its receive status or send status has value one. Obviously, the status flags are used to maintain the system balanced and available.

Example in Fig. 3 shows the closer fog region for Mist Y by dotted line, and the closer region for middle edge node C by the dashed line. In this example, Mist Y receives two service requests from IoT devices. The first request is real- time request, which come from Pacemaker device. This type of request is classified by the Fog Explorer as real-time request. Hence, this request must be handled in the local fog (nod Y). On the contrary, Mist Y is forwarding the soft request to middle edge server C. Also, for the node C the load is migrating to D. This strategy makes the load spread over all system nodes.

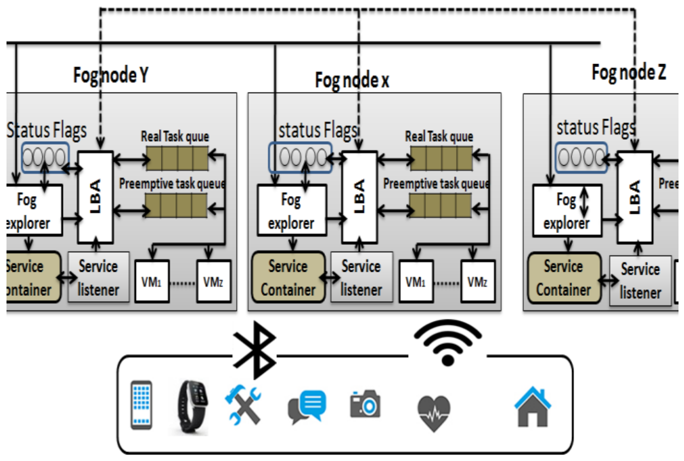

Fig. 2. DLBS Model.

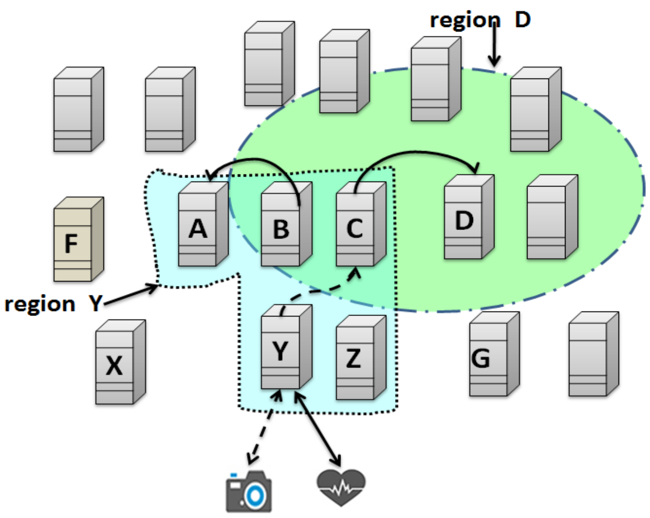

Fig. 3. DLBS Node Region.

A. Load Balancing Allocator (LBA)

The main objective of LBA is to allocate the task requests, which is received by Service Listener (SL). Also, LBA should allocate the real-time tasks to be executed before they met their deadline. Also, it guarantees an efficient response time for the soft tasks. LBA is maintaining to allocate the real-time task trivial waiting time. This accomplished by allocate the real-time task locally or to allocate the task in one of the closed server. In case of soft service is requested, the soft task should be exceeded waiting time threshold (λ). To maintain this condition, the following steps should be computed. First, the total expected execution time of the soft-waiting tasks can be computed as follows.

$$exe_{tot} = \sum_{\forall i} exetime(t_i)$$

Also, the total processing power of the mist node can be formulized as summation of MIPS (million instructions per second processor) for all VMs.

$$p_{tot} = \sum_{\forall j} MIPS(VM_j)$$

Hence, the total expected waiting time should not exceeding λ by achieving the follows equation.

$$w_x = \frac{exe_{tot}}{p_{tot}} + \delta < \lambda$$

Where, δ is the constant depending on the ratio of the real-time tasks μ and the average size of real-time task services ε .

$$\delta = \mu(\frac{\varepsilon}{p_{avg}})$$

Where, p_{avg} is the average of VMs processing power of the mist node. In the following the pseudo code of LBA function is introduced. The following algorithm represents the general steps of load balancing allocator procedure.

Load Balancing Allocator (LBA) Algorithm

Input:

t_k // receive task from the service listener or from other LBA

```
If ($t_k.type = real$)
  freeVM = findIdleVM() // find the idle VM
  If ($idel\_VM \neq \phi$)
    allocateTask(freeVM, $t_k$)
  ElseIf(realTaskLock = 0) // all machines are busy in soft tasks
    freeVM = PreemptSoftTask()
    allocateTask(freeVM, $t_k$)
    If all VM.tasks=real
      realTaskLock = 1
  Else // all VMs are busy in real tasks
    $VM_x = Min_{\forall i}(RemainTime(VM_i))$ /* find a VM with
                                the minimum remaining time */
    $t_k.turnaround = t_k.expectExeTime + VM_x.remainTime()$
    If ($t_k.turnaround \le t_k.deadline$)
      allocateTask($VM_x$, $t_k$)
    Else //find VM in the closest Mist node
      $F_R$ =FogExplorer.getFog(RealTask) //find closest unlock fog for real task
      SendStatusFlag=1
      SendRealTask($F_R, t_k$)
      SendStatusFlag = 0
    End if
  End if
Else //the second case; soft task type
  $exe_{tot} = \sum_{\forall i} exetime(t_i)$ /* Compute the total expected
                execution time of the soft-waiting tasks */
  $w_x = \frac{exe_{tot}}{p_{tot}} + \delta$ // Compute the total expected waiting time
  If($w_x < \lambda$)
    InSoftQueue($t_k$)
  Else
    loadLock=1
    $F_R$ =FogExplorer.getFog(SoftTask) /* find closest unlock
                              fog for soft task*/
    SendStatusFlag=1
    SendSoftTask($F_R, t_k$)
    SendStatusFlag = 0
  End if
```

```
3  End if
```

B. Fog Explorer, Service Container and Flags

Finally, fog explorer module is responsible for determine the closer fog region for each node. This region is defined as set of nodes which has minimum communication overhead. If any of the status flags is change in each node, the node broadcast this information to its closed region. Also, Fog explorer is responsible for broadcasting a copy of the Service Container to all fog and mist computing nodes. Moreover, it should send up-to-date a copy of additional changes in Service Container.

IV. Simulation Setup

As a mist landscape, we propose a fog-mist colony of 100 nodes. Half of the colony nodes are mist nodes, which receive the user requests. The fog-mist colony is connected to a cloud system in circumstance of shortage in the fog-mist sources to the fulfillment of soft task requests. Of these 100 fog-mist colony, 10 are concurrently issuing 1,000 task requests to the mist colony. Furthermore, IoT applications are characterized by two types (real and soft).

The proposed DLBS algorithm, have been implemented on simulator CloudSim [27, 28] 3.0.2 to execute tasks along with Window 7 OS, core i5 2.3 GHz processor and NetBeans IDE 7.2.1. CloudSim computes the execution time of a service request to fulfill a task requirement, hence computes the waiting time for soft task by aggregating the number of instructions necessary to execute the waiting soft tasks. In this experiment, the soft-task request and real-task requests required 0.05, and 0.03 million of instructions per second (mips) respectively. Both task types have 300 MB of incoming and 300 MB of outgoing data. Fog/Mist nodes able to able to handle 250 MIPS. Each fog node can create 10 VM's have the processing power 500 MIPS. The bandwidth between fog nodes is set to 100 Mbit/s, and between the cloud and fog nodes to 10 Mbit/s. All experiments are repeated for 10 times and the mean values are taken.

DLBS model is compared with four models. The first model is FCFS, which serve the tasks based the arrival time. Moreover, the others compared models was created for the cloud computing system, namely the Max-Min, the PBATS and the RETS. The Max-Min maintains a task status table to envision the real loads of the VMs and the evaluated finishing time of tasks, which can distribute the workload among nodes [29]. The Priority Based Autonomic Task Scheduling (PBATS) that schedule its tasks according to three different priorities levels [25, 30]. Furthermore, the Real Time Efficient Scheduling (RETS) depends on reserving a one tenth of the resources for the real-tasks [23]. All these scheduling techniques are matched by the proposed techniques to evaluate the load balancing in the proposed model.

V. Results and Discusion

The performance evaluations have been performed in three dimensions. The first dimension evaluates the performance of the system on the soft-tasks load. On another hand, the second dimension measures the system reliability for the real-time tasks. The performance evaluation based on three parameters, namely; turnaround time, the average waiting time and the throughput. Finally, the third diminution measures the suitability of the model for the real-time services by evaluating the number of failed tasks in the compared algorithms.

This section is organized into three subsections. Each subsection is concerned to evaluate a performance dimension. Hence, the following subsection evaluates the performance of the system using all types of tasks. Moreover, the second subsection evaluates the effect of the system on the real time tasks only. Finally, the failure in the real-service requests is measured in third Subsection.

A. System Performance using Real-Time and Soft Service Requests

In this section three tests are done. The first test measures the response time of variant number of tasks. The second test evaluate the waiting time of the system. Finally, the last test in this section measures the throughput.

1) Turnaround time performance test: The first experiment measure the system performance based on the Turnaround time parameter. DLBS is compared with previous mentioned four algorithms. The experiments are done using different number of workloads from 1000 to 10,000 tasks. The real time tasks will represent 20% from all of the inserted workload in each experiment. Obviously, we can notice that the FCFS curve is rapidly increased by increasing the number of service requests. The bad performance of FCFS is due to the non-preemptive property. Also, Max-Min curve is closed to the FCFS curve. Since, the Max-Min is allocating the longest tasks to VMs which has lest remaining execution time. In another word, in Max-Min scheduling algorithms the short tasks will wait a long time to get the resources, which increase the average of waiting time. In addition, the PBATS curve is keep a less in the average turnaround time results when compared to the FCFS and Max-Min. Indeed, the tasks in the PBATS algorithm are classified into three levels of priorities and underestimate the quality of services. Furthermore, the curve of the RETS refer to acceptable results with a light load up to 1,500 tasks, as shown in Fig. 4(A). Also, RETS gives an inefficient performance if compared by the proposed algorithm (DLBS). The performance of RETS is decreased as increasing the work load. The performance deterioration of RETS algorithm is due to static reservation for the real tasks. It assign one tenth of the resources for the real requests. Reserving a static ratio of the resources can cause problem if there are no proper real tasks. Actually, it is a dilemma if the real tasks exceed the reserved resources. Actually, the DLBS overcome these problems. It gives high priority to the real tasks for satisfy its deadline. Also, it maintains a specific response time for the soft tasks. Subsequently, the DLBS is the most efficient algorithm among all of the compared algorithms in the Mist-fog environment.

2) The waiting time performance test: This experiment measures the waiting time for the service request tasks. As shown in Fig. 4(B), the waiting time of the DLBS curve has the best performance. Moreover, for having a certain QoS the expected waiting time for the soft tasks parameter λ is set by 10 second. Hence, the DLBS curve values are very close to ten second after 5,000 tasks. It is worth noting that the FCFS curve has the worst performance. This bad performance is caused by the same reasons that increase the average turnaround times. Also, the Max-Min curve is the closest one to the FCFS curve. In the PBATS curve the tasks allocation is depending on three levels of priorities, which increase the waiting time for tasks according to their levels. Furthermore, the RETS algorithm has an acceptable performance until the workload less than or equal to 3,000 tasks. Unfortunately, as increasing the services requests, as the average of waiting time is rapidly increased for the RETS. All of these problems have been solved by DLBS algorithm as shown by the performance curve. DLBS maintains an upper bound of the waiting time for each soft task in mist node and send the exceeding load to the closest low load node or to the middle edge node.

3) The throughput performance test: This test measure the performance based on the average of system throughput. The throughput is defined as the total number of finished tasks per time. Additionally, the experiment is done based on the same workload of the past examination. The performance of the compared algorithms is shown in Fig. 4(C). We can notice that, the throughput of DLBS has the best throughputs enhancement compared by the other algorithms. The performance enhancement of DLBS is caused by the balanced distribution of the tasks that satisfy QoS. Also, the worst performance curve is the FCFS. Moreover, the RETS throughput curve is successor to DLBS curve. Since RETS gives the highest priority to the real tasks, which is the lightest processing tasks, then it increases the number of the finished tasks.

B. *System Performance using Real-Time Service Requests*

This experiment evaluates the effect of the proposed system on the real time tasks compared with other algorithms. Each experiment is completed on the real task ratio 25% of workload. Through the experiments, the workloads for all the tasks types are changed from 1,000 to 10,000 tasks. Hence, the real time tasks are changed from 250 to 2,500 tasks. However, all the experiments of the Real-Time tasks are performed in the existence of the soft tasks load. This section is organized as follows. The following subsection measures the turnaround time. Subsection (2) measures the waiting time and Subsection (3) measures the throughput. Finally, the Subsection (4) measures the suitability of the system for real service.

1) Turnaround Time performance Test: The turnaround time performance comparison of the compared algorithms is shown in Fig. 5(A). The worst performance is obtained by the curves that represent the FCFS, PBATS and the Max-Min algorithms respectively. The essential shortage of these algorithms is the disability to handle the requests of the real time service according to their deadline. Actually, these algorithms are not indeed to handle the real-time tasks. Hence, the real times tasks are treated as the soft tasks. On other hand, the RETS gives an acceptable performance when the number of the Real-Tasks are not exceed one tenth of the system resources. As obtained the figure, the performance of RETS result is acceptable until 1,000 real-tasks and is decay after this point. Furthermore, the RETS algorithm preserve the response time of the real time tasks to be less than their deadline times. In other words, the real-time tasks are not presented to any postpone, which limits the turnaround time. Moreover, the real-time tasks are migrated from fog node to another one to avoid waiting time.

2) The waiting time performance test: The averages of waiting time curves that expose the impact of the DLBS model on the duration time of the real tasks are shown in Fig. 5(B). In this figure, the lower mean waiting time is implied for DLBS. As mentioned before, the DLBS model is designed to give the first priority for the Real-Time tasks. Hence, the reserved resources for the soft tasks are released to allocate the real tasks. However, the RETS is the closest curve among all the compared algorithms to the DLBS. Unfortunately, as the real requests load in RETS is increased, as the average waiting time is increased. Hence, the deadline times of the real tasks will be exceeded in RETS model.

3) The throughput performance test: The throughputs curves, in Fig. 5(C), show the performance comparison between the competitive algorithms. Unmistakably, the highest throughput is accomplished by DLBS. The RETS throughput becomes consistent after satisfying the reserved ratio of the real tasks. Also, the DLBS throughput is increased as increasing the real time tasks. Since DLBS algorithm can assign the whole mist node resources and borrows additional resources to satisfy the real-time service requests. Moreover, FCFS has the worst performance because it isn't careless about the deadline of the real-time task.

4) Real-time task failure test: To judge about the suitability of the algorithm for real time services, the task failure should be concerned. To judge about the suitability of the algorithm for real time services, the task failure should be considered. Fig. 5(D) measures real task failure for the proposed and the compared algorithms. The number of task failure for the DLBS model is trivial if compared with the other models. RETS model gives a good performance in low load of the real time tasks. Unfortunately, RETS model doesn't supports flexibility in the reserved resources for the real time tasks. Also, it doesn't support task migration to provide the desired resources. The other algorithms failure values indicate inaptitude for real time services.

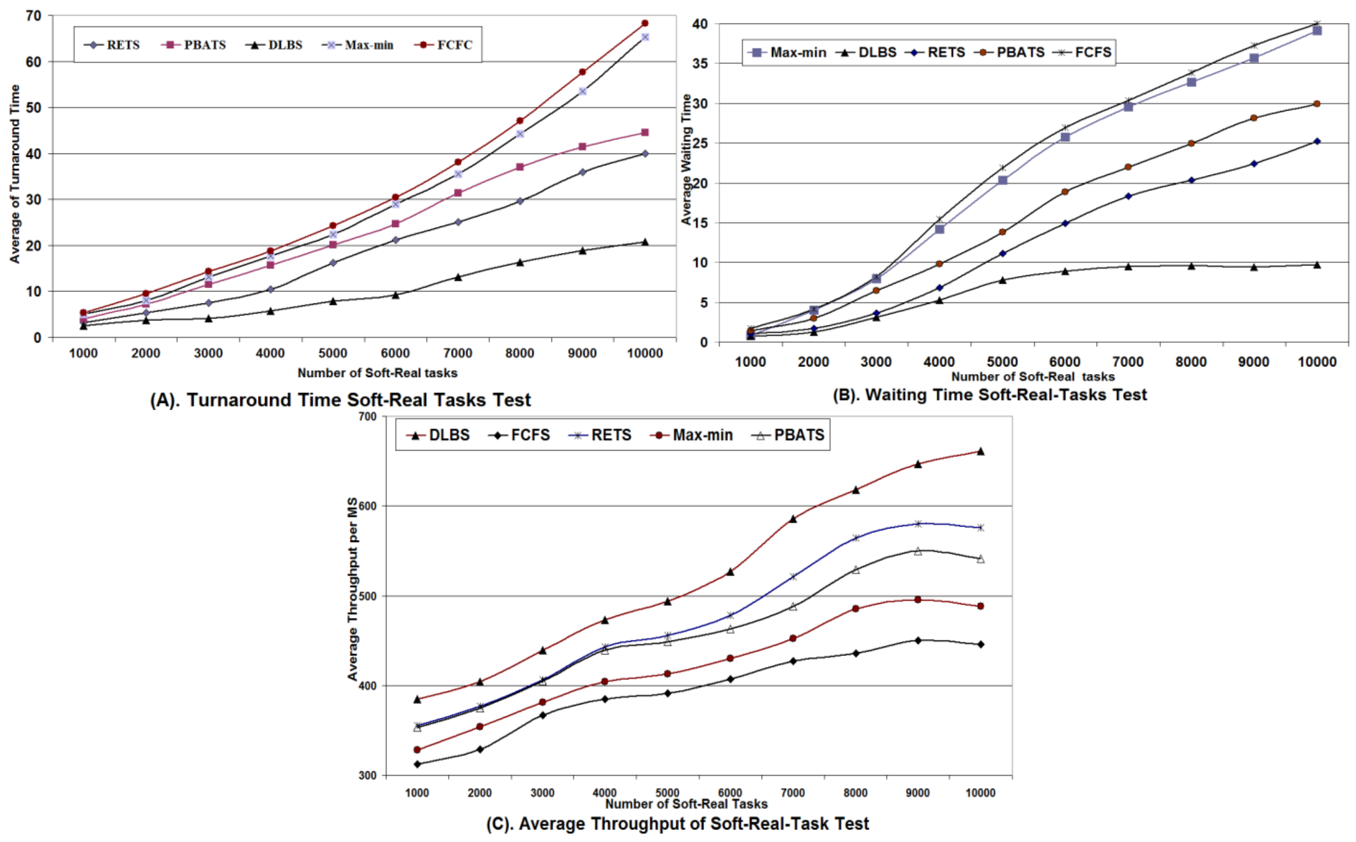

Fig. 4. Performance Comparision using Soft and Real-Time Service Requests.

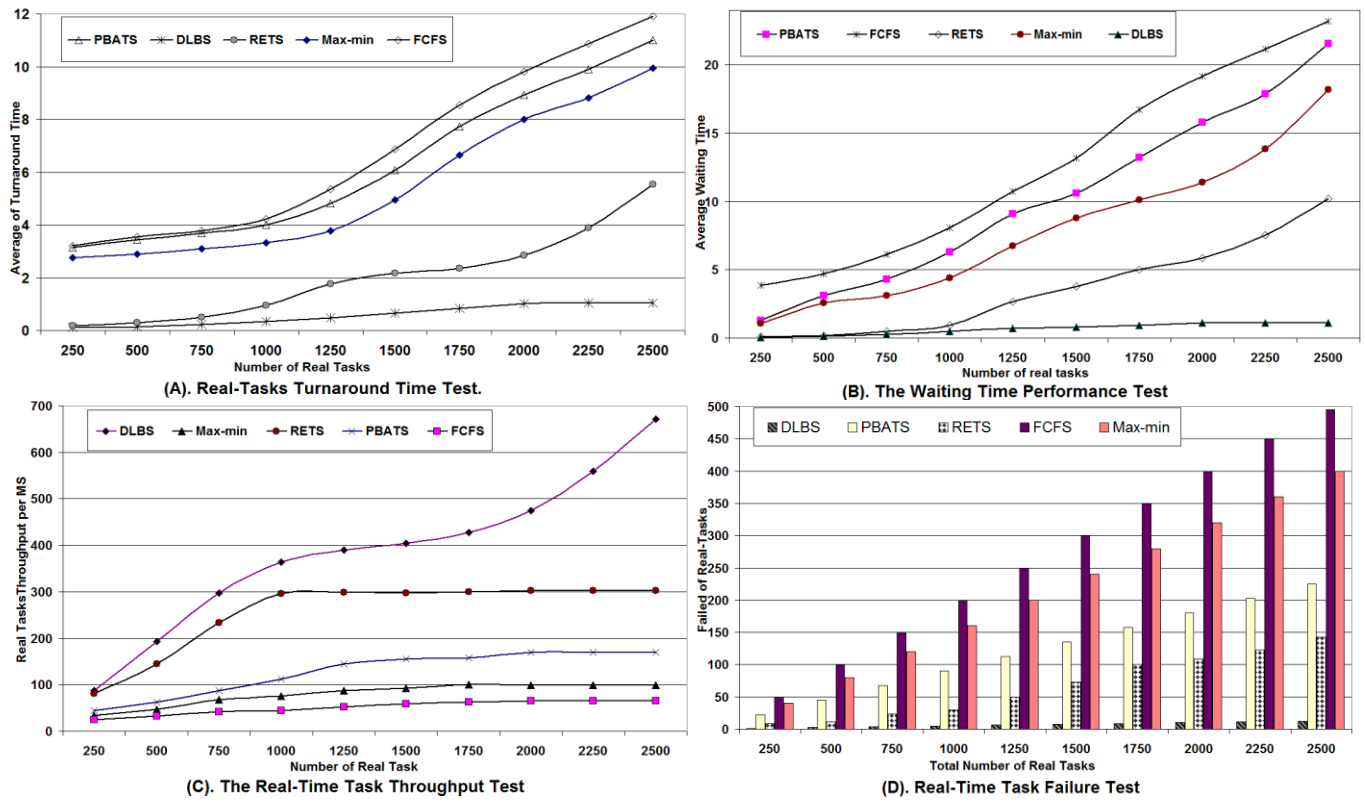

Fig. 5. Real-Tasks Turnaround Time Test.

VI. Conclusion and Future Work

In this paper, DLBS model is designed for managing soft and real time services in fog computing environments. The DLBS model introduces decentralize scheduling algorithm. Fog computing consists of two type of nodes, namely; mist and middle edge nodes. Mist nodes are the closer nodes to IoT devise, which receive its services requests. The DLBS model provides an efficient solution for having IoT service response time. This model is providing an efficient load balancing strategy for IoT service requests. Also, this model manages the IoT services requests load for each fog node in decentralize manner. The decentralize load management avoids the bottleneck problem, which exists in the majority of the other solution. Moreover, this model is designed to fit the real-time serves requests. The experiments show that our methods outperform the compared methods. In future work, this model will be developed to manage the heterogeneous Mist nodes.

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] 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.

[4] 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.

[5] 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.

[6] Mahmood A., Zen H. (2018) Toward Edge-based Caching in Software-defined Heterogeneous Vehicular Networks. In: Mahmood Z. (eds) Fog Computing. Springer, Cham. https://doi.org/10.1007/978-3-319-94890-4_13.

[7] Sari A. (2018). Context-Aware Intelligent Systems for Fog Computing Environments for Cyber-Threat Intelligence. In Fog Computing (pp. 205–225). Cham: Springer. 10.1007/978-3-319-94890-4_10.

[8] Nanxi Chen, Yang Yang, Tao Zhang, Ming-Tuo Zhou, Xiliang Luo, John K. Zao, "Fog as a Service Technology", Communications Magazine IEEE, vol. 56, no. 11, pp. 95-101, 2018.

[9] Luthra, M., Koldehofe, B. & Steinmetz, R. "Transitions for Increased Flexibility in Fog Computing: A Case Study on Complex Event Processing" Informatik Spektrum (2019). https://doi.org/10.1007/s00287-019-01191-0.

[10] [7] A. Davies, Cisco pushes IoT analytics to the extreme edge with mist computing. [Online]. Available: http://rethinkresearch.biz/articles/cisco-pushes-iotanalytics-extreme-edge-mist-computing-2, Blog, Rethink Research.

[11] J.S.Preden, K.Tammemäe, A.Jantsch, M.Leier, A.Riid, E.Calis, The benefits of self-awareness and attention in fog and mist computing, Comput. (Long Beach Calif) 48(7)(2015)37–45.

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

[13] 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.

[14] 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.

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

[16] 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.

[17] W. Lin, C. Zhu, J. Li, B. Liu, and H. Lian, "Novel algorithms and equivalence optimisation for resource allocation in cloud computing,"International Journal of Web and Grid Services,vol. 11, no. 2, pp. 69–78, 2015.

[18] M.Maheswaran,S.Ali,H.J.Siegel,D.Hensgen,andR.F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems,"Journal of Parallel and Distributed Computing, vol. 59, no. 2, pp. 107–131, 1999.

[19] T.D.Brauny,H.Siegely,N.Beckyetal.,"AComparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems,"parallel & distributed computing, vol.61, no.6, pp.810–837,2001.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] Wang J, Li D. Task Scheduling Based on a Hybrid Heuristic Algorithm for Smart Production Line with Fog Computing. Sensors (Basel). 2019;19(5):1023. Published 2019 Feb 28. doi:10.3390/s19051023.

[27] 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.

[28] 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.

[29] 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.

[30] 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.