Informatics Bulletin, Faculty of Computers and Artificial Intelligence, Helwan University
Published Online Vol 4 Issue 2, July 2022
(https://fcihib.journals.ekb.eg)

DLBRT: Dynamic Load-Balance Real-Time Scheduling Model in Fog-Cloud Colony

Hosam E. Refaat
Dept. of Information System
Faculty of Computers and Informatics
Suez Canal University, Egypt.
hosam.refaat@ci.suez.edu.eg

***Abstract*— In recent years, the massive growing of the usage of Internet of Things (IoT) applications expose different challenges in centralized cloud computing paradigm such as; network failure and inadmissible latency for real-time services. The fog layer is added to address these challenges. The nodes in fog computing layer are similar to cloud node, except that it is nearby the IoT devices. Fog node is added to supply IoT devices with the desired resources with low delay. Hence, the performance of IoT application is based on the task scheduling strategy in cloud, or fog computing systems. Hence, the scheduling strategy should maximize resource utilization and increase the resources availability. Most of the previous scheduling methods are suffer from centralization, which consequently represents a performance bottleneck and a single point of failure. Also, some of these scheduling methods ignore the urgency type of the services, which consequently will not be suitable for real-time services. This paper proposes a new load balancing model, which has two main features. The first feature is the dynamic allocation quota for the resource allocation. The second feature is decentralization in fog resource management. In another word, DLBRT manages the IoT service requests based on the service urgency level with maintaining the load among the fog node balanced. The dynamic load balancing strategy in DLBRT is based on redistribution of the less-urgent service requests to the lowest fog node load. In another word, the redistribution of the service requests is depending on the urgency of the real-time service and the workload in every fog. Finally, a simulation model is created to evaluate DLBRT in a fog-cloud colony. Also, the proposed scheduling model is examined with four scheduling models, namely; The FCFS, Max-Min, Real-Time Efficient Scheduling (RETS), and Based Autonomic Task Scheduling (PBATS). We demonstrate through thorough simulations that the performance metrics of turnaround time, waiting time, throughput, and task failure test are enhanced by our suggested approach.**

Keywords*— *Cloud Computing, Fog Computing, IoT, Load balancing, Reliability.

I. INTRODUCTION

Wearable computing, smart metering, smart home/city, connected vehicles, and large-scale wireless sensor networks will all benefit from the fast improvement of computing systems, which will enable the sensing, capturing, aggregation, and processing of streaming data from myriad of connected equipments. These smart things, which are connected to the internet, are known as the Internet of Things (IoT) [1].

A communication infrastructure and computing units are required to implement the Internet of Things concept [2]. IoT storage, connectivity, and computing provisioning are all made possible by cloud computing [3, 4]. It also hides all of the complexities of IoT services and applications. The CloudIoT paradigm refers to the combination of cloud and IoT. This integration aids in the development of new IoT-based apps and services. This integration is discussed in a number of studies [5, 6, 7, 8].

Despite the benefits that the CloudIoT paradigm can provide to large-scale applications, it confronts numerous hurdles [3]. When the number of clients increases, the first obstacle arises. In this case, the requests have been widened in order to enhance the number of services available beyond the cloud's capacity. As the number of client requests grows, so does the time it takes to respond unless the obtainable resources are increased to handle all of the additional tasks. An additional obstacle arises when the IoT-generated data requited a high communication cost to reach the cloud servers. The great distance adds to the difficulty of data security. Furthermore, unpredictably high demand may need the development of a novel load balancing approach. Load balancing is the fair distribution of tasks among parallel resources like networking, hard drives, and computers [9]. In this way, an improvement in the distribution of processing resources and storage devices will be required. To address these issues, Fog computing, a deeply virtualized processing approach, has been demonstrated as a viable solution. CISCO proposes the concept [10] to be used as the cloud edge of an organization. Fog doesn't replace the cloud computing in terms of control. In practice, it serves as a stable domain capable of providing good QoS to a wide range of client requests over short distances. The complete IoT–Fog–Cloud colony is made up of spread fog server nodes and a set of cloud data centers in this fashion. In other words, Fog computing is a cloud computing extension that brings real-time services and low-communication cost to billions of IoT equipment at the network's edge [11, 12]. It is regarded as a virtual platform that

bridges the gap between IoT and cloud computing architecture. It can handle large-scale distributed devices and systems while also supporting their heterogeneity. Consequently, as illustrated in Figure 1, the fog-cloud model reduces network delay and, as a result, energy consumption for all nodes in fog layers [13, 14].

Figure 1 depicts the computational hierarchy that must be leveraged to improve IoT application efficiency. Each request to an IoT application or IoT sensor triggering action can generate one or more tasks. These tasks should be assigned to computation resources based on the urgency of the service while yet maintaining a balanced load. The majority of the prior resource allocation models had an unbalanced load.

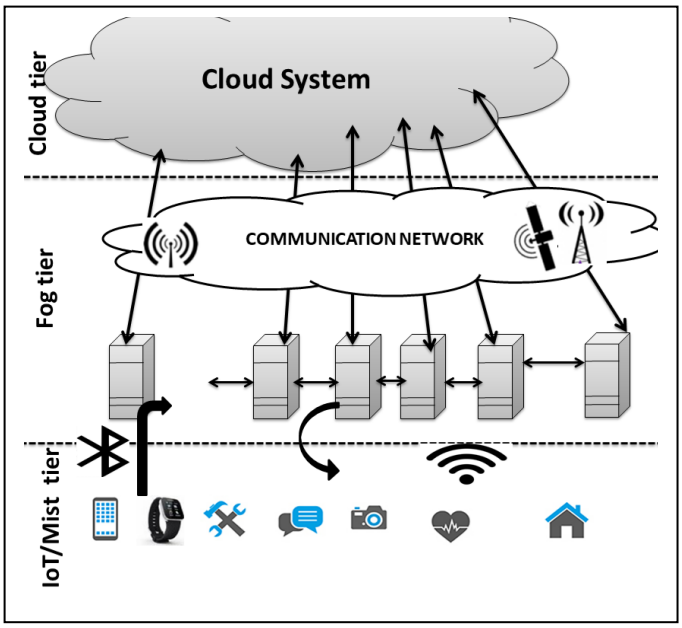


Fig. 1. IoT-Fog-Cloud colony

There are four different sorts of IoT tasks. The first is non-real-time (NRT) tasks, which desired a vast number of resources and large amounts of storage [15]. Tasks of this nature should be sent to the cloud layer. Hence, we will call this type "Cloud Base Task". The Cloud Base tasks haven't dead line but it have specific requirement of Quality-of-Service (QoS) that can be provided by the cloud layer.

Real-time tasks, on the other hand, necessitate a rapid response time. Based on their urgency, real-time services can be categorized based on urgency level as hard, NRT, soft, or firm-real-time [16]. Collecting data from roadside sensors and cameras is an example of this type of task in Intelligent Transportation.

As a result, the second type of task is Soft real-time tasks, which have a deadline but no absolute values, but no system failure or change in results if the deadline is missed. The facial recognition job [17] is an example of this type. The third type, on the other hand, is demanding real-time activities that must adhere to a precise schedule since failing to do so might result in serious problems, accidents, and system failures, such as self-driving car duties. Finally, Firm Real-time Tasks are identical to Hard Real-time Tasks, but a deadline can be permitted to be missed with a small chance of failure. Video conferencing is an example of this type of work, in which conveying data has a deadline with a small chance of being missed, which, if missed, does not cause major problems or failure for the system as a whole.

In general, load balancing appears to be an absolute necessity for scheduling various types of user tasks. The applied state for having a balanced workload among system resources can be static or dynamic. The load in a static task scheduling strategy is dedicated on the processing nodes' recent state, with no regard for future changes. Furthermore, the waiting jobs in this approach are unable to relocate from their processing nodes [18]. Furthermore, static load scheduling solutions don't handle tasks in a proactive pattern. In another hand, the load in a dynamic task scheduling strategy determines job allocation through runtime depend on system state [19]. The dynamic task scheduling methods are used to allocate resources on servers for IoT equipment in order to ensure an adapted distribution of resources. The gratification of justice will shorten the time it takes to complete a task. Furthermore, it will improve job execution speed by utilizing available resources and maximizing storage use to reduce task turnaround time.

In this research, a new framework based on fog-cloud architecture is provided for efficiently managing and executing IoT operations. To address these management issues, the Dynamic Load-Balance Real Time (DLBRT) Scheduling Model in Fog-Cloud Colony model is developed. It prioritizes different sorts of IoT tasks based on their urgency. In addition, the suggested model ensures that the load on the various types of system cluster nodes is evenly distributed. The DLBRT aims to accomplish three things. The primary goal is to serve Real-time tasks according to their type (hard, soft, and firm). The second goal is to create a dynamic resource allocation quota for each type of real-time service, which is based on the current type of service request. The third goal is to keep the workload evenly distributed among the fog nodes.

In the following, the rest of the paper is organized as follows. The next section II; explores the related work of the workload balancing models and mechanisms that are offered for the cloud systems. The proposed model's architecture is detailed in section III. Additionally, the specifics of the model's contained modules are discussed and defined. The performance assessment and the simulation results are obtained in section IV. The paper is concluded in Section V, which also lists the areas for further research.

II. RELATED WORK

Various workload balancing models are presented by different researchers in this section. These models are examined and compared using a variety of criteria, including time limit, processing time, communication cost, service priority, services confidence, scalability, task size, and throughput. In general, cloud technologies have been used to create efficient load balance algorithms. The proposed algorithm is presented in this work and will be implemented in the fog computing environment. This light, on the other hand, is revealed in accordance with their relationship.

The load balancing strategies are generally identical in both cloud nodes and fog nodes, with the exception of one major distinction. Load balancing in fog computing makes balancing operations more practical and operative with finite resources. It provides contact to resources with restricted bandwidth and period. As a result, the fog system fits the requirement of the closed user at a phenomenal value, without the disruption that might come with network traffic.

Thomas et al. [20] introduced the first load balancing approach in this section. This model is purposed to improve service quality by enhancing the usage of the resource based on the job priority and task size. To develop a more stable system, task selection for scheduling can be obtained from both the first and last indexed queues. The jobs are planned using a total credit system based on a grouping of credit size and credit priority determined from task length and task priority, respectively. Finally, the great credit task has a top priority for execution. When the total credits of numerous tasks become equivalent, however, this system has some flaws. In this instance, the FCFS must be deployed with no assurance that jobs will be finished on time or within the specified timeframe.

Mallikarjuna B. et al. [21] suggested a new algorithm based on the comparable behaviour of the Honey bee model (HBB-LB). Priority is used as a primary QoS component in this technique to prevent any process from sitting in the queue for an extended period of time, reducing execution time and increasing throughput. Both types of bees are used in the HBB-LB algorithm. Scout bees are the first category. Its job is to keep looking for food until it finds it. Forager bees receive a signal from scout bees, which defines the second category. Through the waggle/tremble/vibration dance, it creates a pointer to the quantity, quality, and distance from the beehive. When a signal is strong, though, additional foods are available. As a result, the forager bees will follow the scout bees' short journey to the food source.

In the same manner that tasks can be visualized as honey bees, VMs can be regarded as food sources. Furthermore, the VMs are divided into three categories: load balancing, high overload, and low overload. Once the virtual machine become overloaded, the jobs are eliminated and the system behaves like a honey bee. As a result, these tasks are assigned to VMs with low overload. The number of high-priority tasks done on such VMs determines these assignments. It should be mentioned that the VM that has the lowest overload and the fewest executed priority tasks is chosen. Following the proper allocating tasks on the virtual machine, all status data is informed hence the other tasks meets their requirements while the virtual machine is loaded. This model provides a number of benefits, including optimal resource use, and maximizing throughput while maintaining other QoS characteristics based on task priority. Unfortunately, the disadvantages are obtainable at declining priority activities that suffer from extended wait times. These chores could be overlooked, resulting in an unbalanced workload.

Mondala B. et al. [22] presents a dynamic and optimization of a centrally based method for load balancing. The center node makes the distribution decision in this algorithm. The choice is based on the reduced workload associated with sending fewer messages. When the central node fails, however, there is a weakness. In this instance, the system's entire operation will be halted, resulting in system performance degradation. As a result of enhancing the performance, it can be done in two different strategies. The first strategy has been declared as a complete method. As a result, effective values stay assigned to wholly variables in order to achieve the desired results. The answer is ruled out if one of the assigned values proves to be erroneous. The incomplete answer is distinct as the second option, and the main aspect is probability. It accepts that its solution is depend on input factors that produce more accurate results. The feature criteria must provide for problem-solving simplicity, efficacy, and speed. Stochastic Hill Climbing is the name for this method. It is the method of choice for solving the optimization problem. In [14] introduced the Multi-Objective Task Scheduling algorithm, which is based on providing efficient resource utilization to improve throughput.

This technique reduces the execution time in software as a service (SaaS). The service requests are related to the virtual machines in this method in a way that allows for speedier execution. The algorithm is implemented in two stages. Priorities are initially given to the jobs, with the great quality of service being set to a small value and the small quality of service being set to a great value. As a result, tasks with small values are given great priority, and vice versa. Second, the quality of service values has assigned to the VMs in the following manner: high quality of service values are assigned to machines with high processing power, and low quality of service threshold are assigned to machines with low processing power. The arrangement method, on the other hand, is used to organize jobs based on their minimal size and QoS value. From the highest processing power to the lowest processing power, the arrangement method is executed in descending order. The tasks are assigned to a list of sorted machines after sorting is completed. The allocation is done so that the first task in the task list assigns the first machine in the machine list. Similarly, the next task is allocated the second machine in the machine list. Furthermore, the allocation process for the residual machine follows the same pattern. When all of the VMs in the VMs list have been assigned to all of the tasks, the next set of tasks is assigned to the first VM, and the process repeats. The use of the bare minimum of QoS factors, such as execution time, introduces constraints in this approach. As a result, new QoS criteria will need to be included in the future.

An Optimal Priority-Based Service Model [23] introduces the Scheduling Policy. The moral optimization and maximum throughput are provided by this model, which is depending on priority as a resource scheduling approach, entrance management is used. The operations of the model are depending on the provision of full use of the available resources. Its goal is to deliver customer requests faster and spend less time in the queue. The user who pays more than the others, on the other hand, is given priority in using cloud services. The restrictions of the applied features have an impact on the performance of this algorithm. Specifically, futures relating to security and resources hired from other cloud providers.

The architecture for virtualized network load balancing was proposed in [6] to handle the huge data of the costly task scheduling for the network connection. In this case, the load balancer in the data center is set up to modify dynamically based on changes in the data of each user's need and the network service providers' presence. The balancing is carried out on both centralized model. The first load serves as the central, while the second serves as a client that will be chosen in the future by the workload and network traffic balancer. Because of the use of this network traffic balancer, high persistency of web services is achieved. In this architecture, software-type load balancers are preferred over hardware-type load balancers, which impose significant labor and financial costs on customers. In general, some of the benefits have been listed below. The web connection limitation has been removed from the single network. Furthermore, by adjusting the amount of load balancers, the embedded algorithm can be modified by the users. By deploying it in the hybrid cloud, the performance of a large number of internet users could be improved in the future.

In fact, in [17], several scheduling strategies for modifying the employed QoS parameters were presented for varied settings. The scheduling is done in order to generate a large amount of revenue and increase the efficiency of the workload. As a result, there are multiple versions fulfilled for each of the various kinds of scheduling methods, such as Min-Min, first come first service (FCFS), and Max-Min methods. The heuristic strategy, on the other hand, is the most effective. In a cloud computing system, its scheduling processes are divided into three stages. The best aim resource is then chosen. Lastly, the task is sent to the resource that will be used to complete it. Real Efficient Time Scheduling (RETS) has recently been presented in [24]. Its goal is to ensure that real-time tasks are completed without delay. As a result, it preserves ten percent of all available resources for real-time operations. If there are no real-time tasks, however, this proportion is lost.

It's worth mentioning at the end of this section that the Scheduling goal is to reduce response time while fully utilizing resources. As a result, multiple scheduling algorithms based on the deadline have been presented. In [25], the task selection using different tools and in different environments was examined for these methods. These algorithms are built from a variety of perspectives, including processing time cost, delay, response time, and resource consumption time.

III. PROPOSED MODEL

The fog nodes are placed in close proximity to one another. Each fog computing server is centered in a certain place with the primary goal of serving all client requests in that region. As a result, each fog server has its own load balancing method. All customers in the fog computing region require different sorts of real-time (Hard, Firm, Soft) or non-real-time (NRT) operations, and the suggested model is meant to satisfy them. It also introduced the necessary services for all real-time tasks that might be generated by any nearby fog. This situation will arise when one of the neighboring fogs is overloaded by real-time tasks. To address the task constraints, the proposed load balancing approach introduced. The architecture paradigm of the proposed load balancing model over IoT-fog-cloud colony is depicted in Figure 2.

It is made up of three major modules. The first module is the IoT Service Listener (IoTSL), which is responsible for interact with the user/IoT requests. IoT Service Listener (IoTSL) sends each type of the received request task to the appropriate queue. Hence, the task Distributor allocates the received tasks in the VMs based on its deadline and urgency level. The second module is the Load-Distributor-Balancer (LDB). LDB contains two sub-modules, namely; Service Quota Manager (SQM) and Task Distributor. The third module is responsible for interact with the other fog node, which is called "Fog Load Regulator (FLR)". FLR is responsible for forward the excess firm and soft-real-time tasks to the lowest closest fog node. In the event of a fog resource scarcity, the fog colony is connected to a cloud system to ensure that NRT-tasks demands are fulfilled. The next subsection explains in details Load-Distributor-Balancer (LDB) model. Subsection III.A discusses the general algorithm for Fog Load Regulator (FLR).

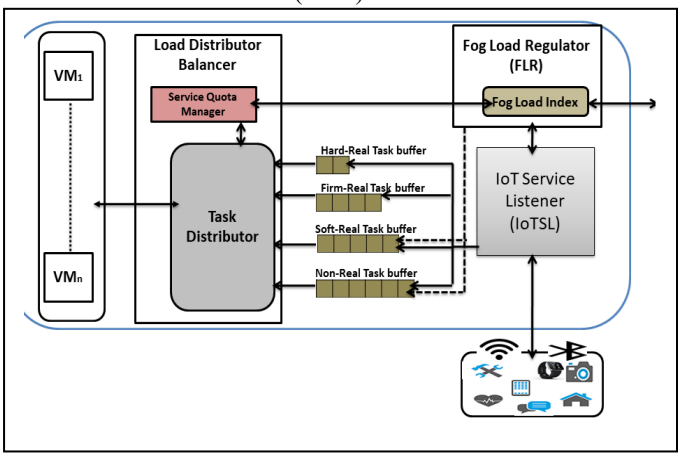


Fig. 2. Dynamic Decentralize Load-Balance Scheduling (DLBRT) Model

III.A) Load-Distributor-Balancer (LDB)

The Load-Distributor-Balancer (LDB) contains two main parts. The first part is to allocate the tasks in VMs of the system, which is called "Task Distributor". This part is discussed in the next subsection. The second part is responsible for divide the resources among the service types.

1) Task Distributor

The main objective of Task Distributor module is to provide the tasks with the resources in efficient performance. Also, Task Distributor should preserve the turnaround time of all types of real-time tasks, to fit their deadline. On another hand, the Task Distributor should maintain in a desired quality of service for the non-real-time task. In terms of resource allocation, the task Distributor gives the highest priority to the hard-real-time task queue. The tasks in the hard-real time queue will be assigned to idle local VMs in the local fog node. If there is no idle are available, task Distributor preempts one of VMs which is currently performing a NRT-task. In the worst case, if all machine are busy with hard-real-time VMs, FRBA compute the expected waiting time if it can wait one of the busy machine. If the deadline does not suffice waiting a local

VM, the neighbor status detector (NSD) find the closest fog node that can execute this task.

Simply, Task Distributor should preserve the turnaround time of all types of real-time tasks, to fit its deadline. On another hand, the FRBA should maintain in a desired quality of service for the NRT-task. It also ensures a fast reaction time for soft, real, time, and NRT-task. For model analysis, Table 1 is used to define the analysis notations.

TABLE I. ANALYSIS NOTATION

Notation	description
η	Turnaround time
ω	The waiting time
$\beta(t)$	The expected executing time for task t .
MIPS(VM)	The speed of the virtual machine VM in million instruction per second
λ_t	The deadline of the task t .
Q_t	The desired quality of service for a task t .
R_H	Resource share-ratio for the hard real-time tasks
R_s	Resource share-ratio for the soft real-time tasks
R_F	Resource share-ratio for the firm real-time tasks
R_N	Resource share-ratio for the non-real-time tasks
R_E	Resource share-ratio for extension of the other quotas
∂	resource adaptation parameter

The turnaround time η for a task t can be computed by the following equation:

$$\eta(t) = \omega + \beta(t) \quad (1)$$

Where ω is the waiting time, and the $\beta(t)$ is the expected executing time for task t .

The waiting time ω , for a task t in queue q , can be computed as summation of the execution time of the previous tasks in the waiting queue, can be computed with the following equation.

$$\omega = \sum_{\forall i \in q} \beta(t_i) \quad (2)$$

The expected waiting time for the task of specific type (for example firm-real-time type) which wait in q queue.

$$\omega = \frac{\sum_{\forall i \in q} exe(t_i)}{\sum_{\forall j} MIPS(VM_j)} \quad (3)$$

In case of different types of real-time tasks the turnaround time η has the following constrain.

The first case is the hard-real-time task, its waiting time must be depressed to zero ($\omega = 0$). Hence, Equation 1 for the hard-real-time can be written as follow.

$$\eta_{hard}(t) \approx \beta(t) \quad (4)$$

The second case is containing the soft and firm-real-time task. The total duration time for these tasks should not exceed the task deadline, as the following equation

$$\eta(t) \le \lambda_t \quad (5)$$

Where λ_t is the deadline of the task t .

The third case is non-real-time task "NRT", it should maintain the following turnaround time condition.

$$\eta(t) \le Q_t \quad (6)$$

Where Q_t is the desired quality of service for a task t .

Resource Balancing Allocator (FRBA) assigns a quota for each type of task. Hence, the resources are divided into five shares, namely; hard-share-ratio, soft-share-ratio, firm-share-ratio, NRT-share-ratio, and extension-share-ratio. Each type of service has a corresponding resource quota, except the extension-share ratio is for stretching the other quota. In another word, the extension-share-ratio is idle resources which allow the other share to stretching to prevent task failure. Each share can be computed as follows:

$$R_H + R_F + R_s + R_N + R_E = 1 \quad (7)$$

After specific time period, each share-ratio will be refined. For example the hared-share-ratio in period *k* is computed based on the previous iteration (*k-1*) and the resource adaptation parameter ∂ , as the following equation.

$$R_{H_k} = R_{H_{k-1}} + \partial \quad (8)$$

$$\partial = \begin{cases} \frac{n}{m}, high\ load \\ -\frac{id}{m}, low\ load \end{cases} \quad (9)$$

Where, *m* is the total number of VMs, n is the number of VMs that taken from the extension area, and *id* is the number of idle VMs in the hard-share-ratio.

Algorithm 1 : Task Distributor

Input:
t // service request generate task by service listener.

```
1.  If ( t.type = Hard ) // case of hard-real-time task
2.    newVM = findIdleVM() // find the idle VM
3.    If ( newVM = ϕ )
4.      newVM = FindLargestDeadline(NRK)
5.      If ( newVM = ϕ )
6.        newVM = FindLargestDeadline(soft)
7.        If ( newVM = ϕ )
8.          newVM = FindLargestDeadline(firm)
9.        End if
10.     End if
11.   End if
12.   HardVM++ /* Increase number of allocated VM
```

for hard-real-time */
13. allocate (newVM, t)
14. If (HardV$M > R_{H_{k-1}}$) /* number VM allocated for the hard- real-time exceed the hard-quota */
15. R_{H_k} = shareRatio($HardVM$)
16 Else if ($t.type = Firm$) // case of firm-real-time task
17. newVM = findIdleVM() // find the idle VM
18. If (newVM = ϕ)
19. newVM = FindLargestDeadline(NRK)
20. If (newVM = ϕ)
21. newVM = FindLargestDeadline(soft)
22. End if
23. End if
24. FirmVM++ /* Increase number of allocated VM for firm-real-time */
25. allocate (newVM, t)
26. If (FirmV$M > R_{f_{k-1}}$) /* number VM allocated for the firm-real-time exceed the firm-quota */
27. R_{f_k} = shareRatio($firmVM$)
28 Else if ($t.type = soft$)// case of soft-real-time task
29. newVM = findIdleVM() // find the idle VM
30. If ((newVM = ϕ)&(Soft_Lock=0))
31. newVM = FindLargestDeadline(NRK)
32. If (newVM ≠ ϕ)
33. allocate (newVM, t)
34. If (softV$M > R_{s_{k-1}}$) /* number VM allocated for the soft-real-time exceed the soft-quota */
35. R_{H_k} = shareRatio($softVM$)
36. End if
37. End if
38. Else // there is lack in resources
39. fog_x = FindNearestFog() /*find closest fog node with low load*/
40. $send(fog_x, t)$ // send the task t to fog x
41 Else // the case of non-real-time task
42. If ($t.q < Q$)
43. newVM = findIdleVM() // find the idle VM
44. If (newVM ≠ ϕ)
45. NRTVM++ /* Increase number of allocated VM for Non-real-time tasks*/
46. allocate (newVM, t)
47. Else // no idle VM
48. $send(cloud, t)$ // send the task t to cloud
49. End if
50. End if

2) Service quota manager (SQM)

The service Quota manager (SQM) quantifies the system resources into five virtual portions, namely; hard-share-ratio, soft-share-ratio, firm-share-ratio, NRT-share-ratio, and extension-share-ratio. Each portion is dedicated to a specific type of service. SQM has two main objectives. The first objective is maintaining the resources available in demand the high urgency services (hard and firm real-time services). This objective is achieved by controlling the resource reservation by the soft and NRT service without causing service failure. The second objective is to achieve adaptive resources division for each service type based on the current load status.

The service Quota manager (SQM) is process triggered periodically to evaluate the current status of the system usability. In another word, at a specific period of time θ the SQM re-evaluate the quota of each type of service. The time period θ is considered as the average size of the hard task. The general form of the service quota is obtained in Equation 5. Each service type quota can be computed as a ration of the actual resource usage. The following equation computes the hard-share-ratio.

$$R_H = \frac{\sum_{i \in hard} exe(t_i)}{\sum_{\forall j} MIPS(VM_j)} \quad (10)$$

Consequently, the firm-share-ratio can be computed as follow.

$$R_F = \frac{\sum_{i \in firm} exe(t_i)}{\sum_{\forall j} MIPS(VM_j)} \quad (11)$$

The R_E extension -share-ratio is considered as constant value, which provides the system with ready machine for the urgent service. Hence, the R_N NRT-share-ratio can be deduced by the following equation.

$$R_s = 1 - (R_H + R_F + R_N + R_E) \quad (12)$$

The service Quota manager (SQM) method is shown the following algorithm.

Algorithm 2: Service quota manager

51 For each time period θ

52. $$R_{H_k} = \frac{\sum_{i \in hard} exe(t_i)}{\sum_{\forall j} MIPS(VM_j)}$$

53. $$R_{F_k} = \frac{\sum_{i \in firm} exe(t_i)}{\sum_{\forall j} MIPS(VM_j)}$$

54. $$R_s = 1 - (R_H + R_F + R_N + R_E)$$

// where R_E is consistent

55. If (VMnumber(Rs) $\geq SoftVM$)

56. Soft_Lock=1

57. If (VMnumber(R_N) $\geq NRTVM$)

58. NRT_Lock=1

59. End for

III.B) Fog Load Regulator (FLR)

The tasks in the Hard-real-time queue will have high priority and will be allocated to one of the idle local VMs in the node if exist. Task Distributor preempts one of the soft-real-time or NRT-task in busy VMs if there is no idle. If there are no idle virtual machines or all virtual machines are occupied, the Fog Load Regulator (FLR) chooses the least cast resources at the closest Fog node in the worst-case scenario. The Fog Load Regulator (FLR) obtains the Fog-Load-Index to determine the status of the other fog node. Each fog broadcast is status periodically to the other fog nodes in its close region, as shown in Figure 3. The example in Figure 3 displays the fog region for fog Y by dotted line. Each fog node determines its closed region. Hence, Fog Load Regulator (FLR) in fog Y collects all information about it closed fog node and register these data in its Fog Load Index. Fog Load Index contains average of the work-load and two of status flags. The status flags are called firm-lock and soft-lock, which decide the capability to receive firm-real-task and soft-real-task form the other nodes.

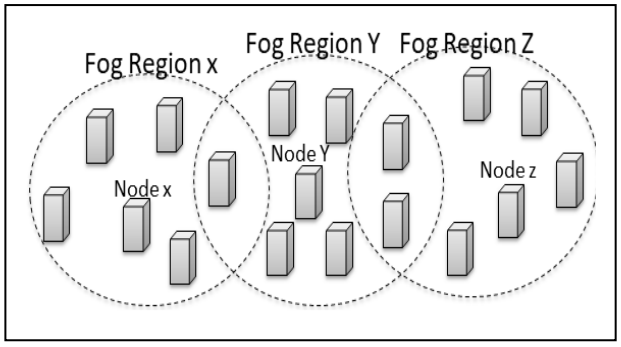


Fig. 3. Fog Neighbor Reagon for Load Regulator example

If the expected waiting time exceeds the QoS threshold (λ), Soft-Lock, which is a lock for soft-real-time task waiting, is set to zero. Soft-Lock is set by one if the expected waiting time overtake QoS threshold (λ), where λ is the soft-real-time task's average deadline. Also, the Firm-Lock is set by one if the number of allocated VMs by the Firm tasks exceeds the Firm-Quota. Any fog node cannot receive a firm/soft-real task if it's lock have value 1. Moreover, if all VMs are allocated by Soft/Firm-real-time tasks, the Soft/Firm-real-time task lock is set to one.

IV. SIMULATION SETUP AND RESULTS

WorkflowSim [26] is investigated to simulate the various scheduling approaches in order to analyses the experimental outcomes. The WorkflowSim [26] is an open-source modeling workload that builds on the CloudSim. In the fogs computations, the simulation assessment is done by employing homogenous features. They're based on the characteristics of Amazon EC2 virtual machines. As a result, every test is carried out on a free T2.Micro instance of Amazon EC2.

The proposed model was used to compare the results of four different models. To begin, there is the FCFS, which is used to perform the tasks based on their interning time. In addition, three more comparative models for the fog/cloud environment have already been published. The Max-Min, PBATS, and RETS are the three. The Max-Min maintains a task status table to estimate actual VM loads and task turnaround time, allowing the workload to be distributed between VMs [22]. Priority Based Autonomic Task Scheduling (PBATS) is a system that schedules task consist three levels of priority [27]. Furthermore, the Real-Time Efficient Scheduling (RETS) model is depending on devoting a 10 percent of the time to real-tasks [25]. The proposed method is matched with these scheduling strategies in order to assess best performance and best resource usage in the given model.

The assessments of performance were carried out in two aspects. The first component assesses performance in terms of several types of task urgency. The turnaround time, average waiting time, and throughput are the three parameters used to evaluate performance. Finally, the second diminution evaluates the amount of failed tasks in the compared algorithms to determine the model's fitness for hard, soft, and firm-real-time services. Five times each experiment was carried out. The average of the five runs was taken and used for test comparison.

A) System Performance

In the next subsection, the response time measurement is done. Moreover, subsection II evaluate the waiting time for the task in the compared models. Finally, subsection III compares the throughput in the models that were compared.

A.1) Turnaround Time test

This test measures the performance focus on the Turnaround parameter. The proposed model (DLBRT) is contrasted with four models listed above. Every test is carried out with 10 diverse workloads ranging from 40 to 400 tasks. The hard-real-time tasks will account for 20% of the total workload added in each trial. All of the tasks are run on each machine's 10 VMs with a total processing power of 2000 MIPS. The acquired findings are displayed, which indicate the average turnaround times of all methods in Figure 4.

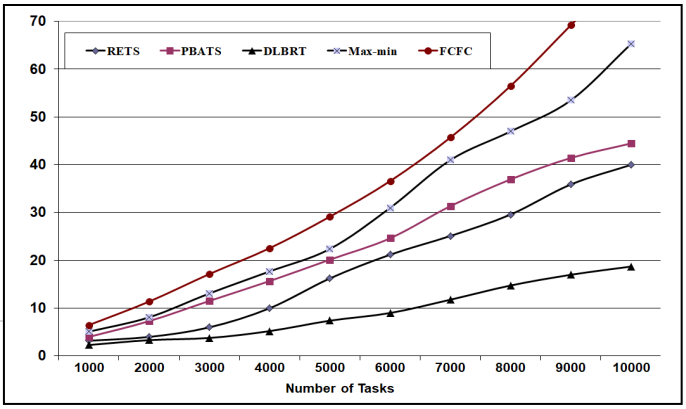
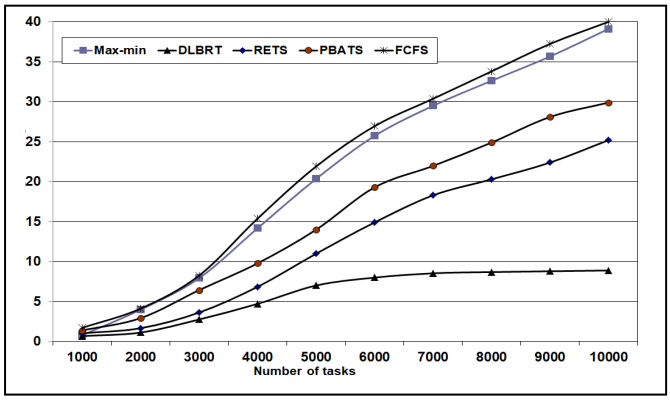


Fig. 4. Turnaround Times Comparison

Clearly, we can notice that the FCFS model has quickly increased as increasing the number of tasks. The highest turnaround time is for FCFS on account of non-preemptive property. Also, Max-Min model has performance is near to the FCFS curve. Since, the Max-Min dedicates the highest execution time tasks to VMs has the least remaining execution time. In another word, in Max-Min model Leads the soft/firm-real-time tasks, which mostly have a short size, will wait a long time to get the resources. This strategy will increase the average waiting time. Moreover, the PBATS model has little enhancement compared to the previous models. Certainly, PBATS model has three levels of task priority with despising QoS. Moreover, the RETS curve has good performance in low workload, as shown in Figure 5. Unfortunately, the performance of RETS model is deteriorated as increasing the workload. The performance deterioration of RETS model is caused by static allocation for the real tasks. Moreover, RETS model doesn't distinguish between the different types of real-time tasks. Actually, the shortages in the previous models are overcome by DLBRT model. DLBRT assigns a different priority for the tasks based on its urgency. In case of high workload DLBRT, it reduces the load by forward the NRT-tasks to cloud and soft-real-time tasks to the closet low-load node. Hence, DLBRT can control the workload on the fog nodes to fit each task requirement. Subsequently, the DLBRT has the most effective performance through the compared



model.

Fig. 5. The Waiting Time Comparison

A.2) The Waiting Time Test

This test evaluate the expected waiting time by the compared model. Figure 5 depicts the waiting time cost in each model. Obviously, the DLBRT has the least waiting time. Not to mention, the FCFS model gives the highest waiting time cost. Also, the waiting time for the Max-Min curve is much closed to FCFS. It should be noted that FCFS model gives the highest waiting time cost. The deterioration of the FCFS and Max-Min performance curves is caused by the high cast of the waiting time. In the PBATS curve, the task allocation is based on priority level, which gives a little performance enhancement, but it hasn't a load-balancing among the fog nodes. Furthermore, unless the load is less than or equal to 3,000 tasks, the RETS model performs well. Regrettably, as the number of tasks grows, so does the average waiting time for RETS. As indicated by the performance curve, the DLBRT algorithm was able to tackle all of these difficulties by two main strategies. The first strategy is based on reducing the number of NRT tasks by sending most of these tasks to the cloud servers. Since the NRT tasks contain massive computations, then the execution of NRT tasks in fog nodes increases the waiting time intensely. The second strategy is based on redistributing the load between the fog nodes, which reduces the number of waiting tasks in each fog node.

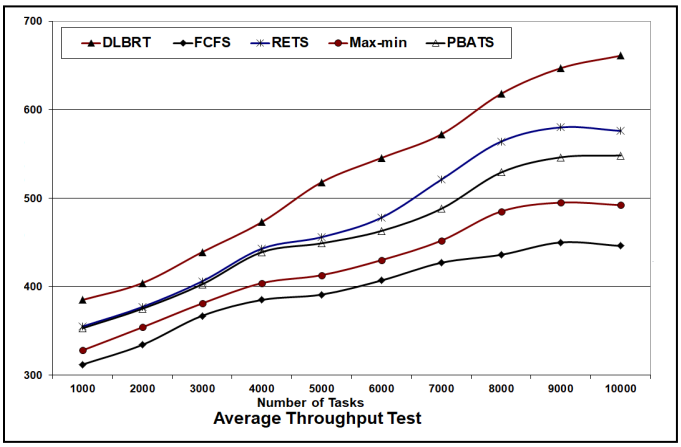
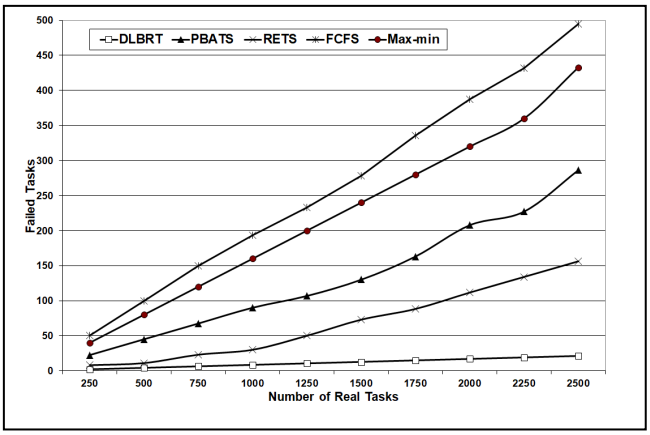


Fig. 6. The Throughput Performance Comparison

A.3) The Throughput Performance Test

The average system throughput is used to measure performance in this test. The throughput is the total number of completed tasks per unit of time. Furthermore, the experiment is carried out with the same workload as the previous examination. Figure 6 depicts the performance of the comparing methods. When compared to the other methods, we can see that DLBRT has the best throughput improvement. The reduction of the waiting time, as discussed in the previous test, increases the system throughput. Moreover, the balanced



distribution of jobs that fulfill QoS is what causes DLBRT to function better. The FCFS is also the weakest performance curve.

Fig. 7. Task Failure test Comparison

B) Task Failure Test

The task failure should be addressed while evaluating the algorithm's applicability for real-time services. This section discusses the suitability of the compared models. As mentioned before, there are three types of real-time tasks. In this experiment, the ratio of hard, firm, and soft-real-time tasks is 2, 8, and 10 respectively. Figure 7 compares and contrasts the proposed and evaluated algorithms in terms of real-time task failure. When compared to the other models, the number of task failures for the DLBRT model is negligible. Since DLBRT System is based on the processing of tasks based on their urgency, the task failure chances are low. The RETS model performs well under low demand in real-time operations. Regrettably, the RETS architecture does not provide for adaptability in resource allocation for real-time operations. Furthermore, task migration is not supported in order to supply the needed resources. The failure values of the other algorithms show inadequacy for real-time services.

V. CONCLUSION AND FUTURE WORK

In this paper, the DLBRT model is introduced as a multi-priority level scheduling that achieves decentralized workload balancing in the IoT-Fog-cloud colony. The model was created to allow for successful fog resource allocation in a variety of NRT, soft, firm, and hard-real-time applications. A new dynamic resource allocation has been provided by the suggested methodology, which is based on the variation of the workload type. The proposed model has allowed a flexible allocation of real-time tasks based on their deadlines and urgency levels. Moreover, the NRT tasks are buffered in a queue to be allocated in the local fog in case of enough available resources. In another word, the exceeded load in the local fog server will be transferred to the nearest server in the fog colony or sent to the cloud to be provided by the DLBRT. We have two goals for the future work. The first goal is re-engineering the system for making it more suitable for large-scale areas. The second goal is to improve this model so that it can take advantage of the benefits of heterogeneous processing power.

REFERENCES

[1] Giannelli, C.; Picone, M. Editorial “Industrial IoT as IT and OT Convergence: Challenges and Opportunities”. IoT 2022, vol. 3, pp. 259-261. 2022. https://doi.org/10.3390/iot3010014

[2] Pradeep, P.; Kant, K. Conflict Detection and Resolution in IoT Systems: A Survey. IoT 2022, vol. 3, pp. 191-218. 2022 https://doi.org/10.3390/iot3010012

[3] Gigli, M. and Koo, S. Internet of Things, Services and Applications Categorization. Advances in Internet of Things, vol. 1, pp. 27-31. 2011. http://dx.doi.org/10.4236/ait.2011.12004

[4] Odun-Ayo, Isaac & Okereke, Chinonso & Evwieroghene, Orovwode. (2018). Cloud Computing and Internet of Things - Issues and Developments. Proceedings of the World Congress on Engineering 2018 Vol I

[5] Navadia, Nipun R.,et al. "Applications of Cloud-Based Internet of Things." Integration and Implementation of the Internet of Things Through Cloud Computing, edited by Pradeep Tomar, IGI Global, pp. 65-84, 2021. https://doi.org/10.4018/978-1-7998-6981-8.ch004

[6] Tabrizi, Sahar & Ibrahim, dogan. A Review on Cloud Computing and Internet of Things. International Journal of Computer, Electrical, Automation, Control and Information Engineering. Vol. 11. pp. 462-467, 2017.

[7] Mondragón-Ruiz, G., Tenorio-Trigoso, A., Castillo-Cara, M. et al. An experimental study of fog and cloud computing in CEP-based Real-Time IoT applications. J Cloud Comp vol 32, no. 10, 2021. https://doi.org/10.1186/s13677-021-00245-7

[8] Niedermayer, H., Holz, R., Pahl, M.-O., Carle, G., 2010. On using home net-works and cloud computing for a future internet of things. In: Future Internet-FIS 2009. Springer, pp. 70–80.

[9] Sarkar, S., Misra, S., “Theoretical modelling of fog computing: a green computing paradigm to support iot applications”, IET Networks 5(2) (2016) 23–29

[10] MarketWatch: ‘Cisco delivers vision of fog computing to accelerate value from billions of connected devices’, available at http://www.theiet.org/resources/ journals/research/index.cfm, accessed August 2014

[11] Hong, K., Lillethun, D., Ramachandran, U., et al.: ‘Mobile fog: A program-ming model for large-scale applications on the internet of things’. Proc. of the Se-cond ACM SIGCOMM Workshop on Mobile Cloud Computing, Hong Kong, China, August 2013, pp. 15–20

[12] Stolfo, S.F., Salem, M.B., Keromytis, A.D.: ‘Fog computing: Mitigating insider data theft attacks in the cloud’. IEEE Symp. on Security and Privacy Workshops, San Francisco, USA, May 2012, pp. 125–128

[13] Preden, J.S., Tammemae, K., Jantsch, A., et al.: ‘The benefits of self-awareness and attention in fog and mist computing’, Comput. Mag., 2015, 48, (7), pp. 37–45

[14] Javed, Waheed & Parveen, Gulnaz & Aabid, Fatima & Rubab, Syeda & Ikram, Sidra & Rehman, Khawaja Ubaid & Danish, Muhammad. A Review on Fog Computing for the Internet of Things. vol. 10, pp. 1-7, 2021. 10.1109/ICIC53490.2021.9692966.

[15] Bonomi, F., Milito, R., Natarajan, P., et al.: ‘Fog Computing: A platform for internet of things and analytics’, in Bessis, N., Dobre, C. (Eds.): ‘Big data and in-ternet of things: a roadmap for smart environments – part I’ (Springer International Publishing, Switzerland, 2014), vol. 546, pp. 169–186

[16] J.S. Preden, K. Tammemäe, A. Jantsch, M. Leier, A. Riid, E. Calis, The bene-fits of self-awareness and attention in fog and mist computing. Computer 48 (7), 37–45 (2015).

[17] Manas Kumar Yogi, K. Chandrasekhar, G. Vijay Kumar - Mist Computing: Principles, Trends and Future Direction, SSRG International Journal of Computer Science and Engineering (SSRG-IJCSE) – volume 4 Issue 7 – July 2017

[18] Mihai, Viorel et al. “WSN and Fog Computing Integration for Intelligent Data Processing.” 2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI) (2018): 1-4.

[19] Asif-Ur-Rahman, Md. et al. “Toward a Heterogeneous Mist, Fog, and Cloud-Based Framework for the Internet of Healthcare Things.” IEEE Internet of Things Journal 6 (2019): 4049-4062.

[20] A. Thomasa, Krishnalal Ga, Jagathy Raj V Pb, “Credit Based Scheduling Agorithm in Cloud Computing Environment”, ICICT 2014 pp. 913 – 920 2015.

[21] B. Mallikarjuna & Vankara, Jayavani & Sujatha, V. (2015). Honey Bee Behaviour imitates the Artificial Algorithm for Load Balancing of tasks in Cloud computing. International Journal of Applied Engineering Research. 10. 177-182.

[22] 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.

[23] Guruprasad, H S & .M, Dakshayini. (2011). An Optimal Model for Priority based Service Scheduling Policy for Cloud Computing Environment. International Journal of Computer Applications. 32. 23-29.

[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

[25] 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

[26] 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

[27] Bala, Anju & Chana, Inderveer. Multilevel Priority-Based Task Scheduling Algorithm for Workflows in Cloud Computing Environment. vol 408. pp. 685-693, 2016. DOI:10.1007/978-981-10-0129-1_71.

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, Edige Computing, and Datamining.