Informatics Bulletin, Faculty of Computers and Information, Helwan University,
Published Online ***Vol 2 Issue 1, January 2020***
(http://fcihib.fci.helwan.edu.eg)

MLITS:Multi-Level tasks scheduling model for IoT Service Provisioning

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

***Abstract*— In Internet of Things (IoT) environment there are a huge number of devices that need to communicate and send data continuously between these devices also between them and the cloud data center. This data is increasing exponentially, exposing the IoT environment to collapse. Therefore, so we need an environment that supports the stability of the IoT and at the same time increases the speed of exchange huge data between IoT devices. Fog computing and mist computing support real time data collection and analysis locally at IoT devices. In addition, fog computing and mist computing overcome on various challenges like network bandwidth, and reliability. Resource management and allocation for IoT tasks in three levels mist-fog-cloud architecture suffer from a lack of approaches and frameworks that handle this situation in an efficient manner. In order to address this shortage, Multi-Level IoT Tasks Scheduling (MLITS) model is introduced in this paper. MLITS is an orchestration system for managing and allocating IoT tasks over the mist-fog-cloud architecture. The proposed model performs IoT tasks based on their deadline and the urgency of their execution. In addition, it performs various types of IoT tasks without rapidly consuming available sources. Moreover, the proposed model maintains the resource usage balanced. Finally, The MLITS is simulated and evaluated on truthful fog resources and various workload circumstances. Also, the proposed scheduling model is compared with three scheduling model, namely; Min-Min, Credit-Based-Scheduling (CBS) and Earliest-Feasible-Deadline-First (EFDF). Through extensive simulations, we show that our proposed model enhances the performance metrics, namely; turnaround time, waiting time and throughput.**

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

I. INTRODUCTION

The idea of the IoT emerged from the fact that most devices and humans are connected most of the time to the Internet [1]. The IoT is a modern technique that is designed to allow all types of devices including tools, sensors, various Artificial Intelligent (AI) tools and more to communicate data between them in a secure manner [2]. To implement the concept of Internet of Things we need a communication infrastructure, and computational units [3]. Cloud computing introduces storage resources, communication and computing provisioning for IoT [4]. Furthermore, it hides all complexity of IoT services and applications. The integration of cloud and IoT is called CloudIoT paradigm. This integration helps to provide new types of applications and services based on IoT. There are various works that introduces this integration [5, 6, 7, 8].

Although this integration has been successful in many cases, it suffers from some shortcomings of presence of thousands of billion IoT devices that generate huge data and needs real-time analysis [9, 10, 11]. The implementation of large IoT devices and services is increasing the service latency by using cloud computing in unacceptable manner. Moreover, the energy consumption to transfer data is high. Therefore, it affects the consumption of batteries of IoT devices significantly, which we need to maintain its capacity for a long time. More broadly, the use of cloud computing alone suffers from the fact that the (IOT) is vulnerable to hacking and loss of security. Fog computing architecture gives a reliable solution to solve most of the problems associated with using Cloud computing architecture [9, 12, 13].

Fog computing is an extension for the cloud computing, it introduces a real-time and low latency services to billions of IoT devices at the edge of the network [14, 15]. It considered as virtual platform between IoT and cloud computing architecture. It can handle large-scale distributed devices and systems and support the heterogeneity of these devices and systems. Fog computing environment contains two types of nodes. The first type is the extreme edge node which is also called mist node. The mist computing facilitates the deployment of IoT services with its combination with IoT devices [16, 17]. It brings the computation closer to sensors and actuators in IoT environment. The second type is middle fog nodes which represent additional resources in the fog system and far from the IoT devices. Also, the mist computing is integrated with the fog computing and cloud computing concepts to enhance the data collection and processing in IoT environment. Therefore, the mist-fog-cloud colony will reduce the network delay and consequently the energy consuming for all nodes in mist, and fog, layers [18, 19], as shown in Figure 1.

IoT tasks can be classified into four types, real-time tasks that have less computation density, real-time tasks that have high computation density, Non-real time tasks that have specific requirement of Quality-of-Service (QoS), and Non-real time tasks that required massive amount of resources and huge-volume storage [20].

In this paper a new framework based on mist-fog-cloud architecture is proposed to manage and execute IoT tasks in an efficiency manner. **M**ulti-**L**evel-**I**oT-**T**ask-**S**cheduling (MLITS) model is proposed to overcome these challenges of management. It allocates the various types of IoT tasks according to their urgency. Also, the proposed model maintains a balanced load among the different types of system cluster nodes. Besides, this model Maintains energy consumption rates for IoT devices, mist and fog nodes.

II. RELATED WORK

In this paper, analysis of various research in the field of Mist computing, fog computing, cloud computing and IoT integration are introduced. Fog computing frameworks are able to facilitate all IoT needs [21]. Resource management, IoT devices communication, all of them can be management by fog computing frameworks. In [21], Sarkar introduces a theoretical model of fog computing framework that can be integrated with IoT. He measures the service latency and energy consumption compared to cloud computing approach. Also, Rahman et.al in [19] introduces Internet-of-Healthcare-Things (IoHT) framework. Unfortunately, this framework is based on static load balancing. There are various fog computing frameworks that can be integrated with IoT to provide many tools to manage IoT services [22, 23].

A Distributed Dataflow programming model is proposed in [22] to manage IoT application in Fog computing environment. In this framework the most requirements to build IoT application are determined and processed. However, no measure of performance has been provided to measure the effectiveness of this framework. In [23] a new fog computing framework to resource allocation for IoT is proposed. In this framework the latency is reduced, and the fault tolerance and privacy are considered.

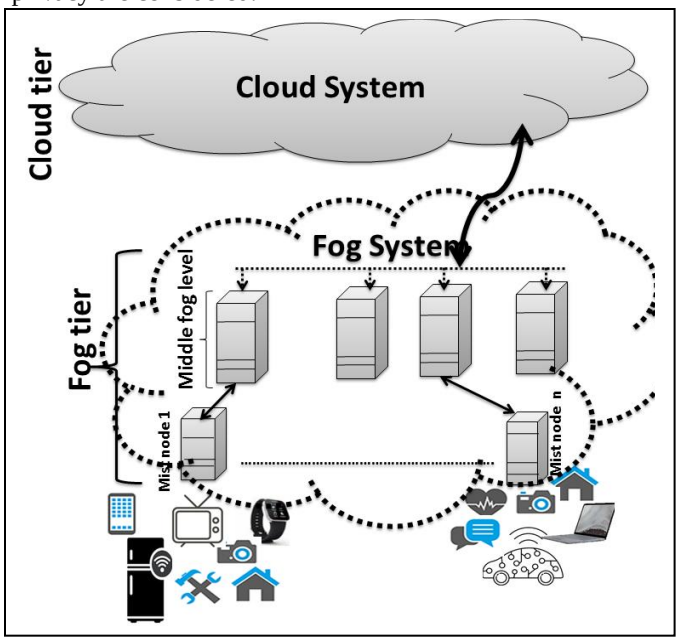


Fig. 1. IoT with Mist-Fog-Cloud colony

Mist computing enhances the communication and computing availability for all nodes and IoT devices. Cloud, fog, and mist computing work together to introduce the Mist-Fog-Cloud architecture [19, 24] that enhances the IoT environment. The proposed work in this paper deals with this new architecture to capture its advantage that fit IoT environment.

There are various frameworks that which are similar to our work. Min-Min Algorithm, Credit-Based-Scheduling (CBS) Algorithm, Improved Priority based Job Scheduling Algorithm, and Earliest-Feasible-Deadline-First (EFDF) are four research that try to allocate IoT tasks in an efficiency way in cloud computing environment. These researches are used to evaluate the proposed MLITS model.

In [25] the Min-Min Algorithm for task scheduling is introduced. It is based on minimum completion time (MCT) for tasks to allocate them to resources. In this algorithm, the task that has less time to use the resources is allocated firstly.

In [26] the Credit-Based-Scheduling (CBS) Algorithm is proposed. CBS algorithm concentrates on tasks criteria, priority and length, to management their scheduling. The task deadline did not implement in this work and left for the future. In [27] an improvement in priority-based job scheduling algorithm is introduced. The task priority is considered the major factor in this research to enhance the makespan for tasks scheduling. The EFDF algorithm is introduced in [28]. The tasks deadline is main attribute to schedule them in this algorithm. It arranges all tasks in a queue based on their closest to its deadline, then pick up one of them to be allocated. Finally, all of these algorithms Suffer from the limited factors that deal with to schedule tasks. In another word, these algorithms are focus on a specific scheduling criterion such as priority and neglecting the other criteria such as; task size, and deadline.

III. PROPOSED MODEL

The proposed model contains three tiers: IoT, fog and cloud. The first tier is IoT which sometimes is also referred to as "IoT &dew". This tier is responsible for interact with the world. IoT includes the sensors for collecting the data from the environment, and the actors which have an effect on it. Dew is a solution for real time IoT applications, which required computation resources with a negligible delay. Unfortunately, Dew supports limited computing power and storage resources. The second layer is the fog tier, which is divided into two sub-layers; mist and middle fog level. The mist layer provides dedicated nodes, which can be accessible with low communication overhead. In another word, the mist nodes are closed to IoT devices. On the other hand, the middle fog layer contains fog nodes and orchestration node. The orchestration node is responsible for managing the load among the mist nodes and middle fog nodes. The last layer is cloud computing, which provides boundless amount of computing nodes, and huge-volume storage.

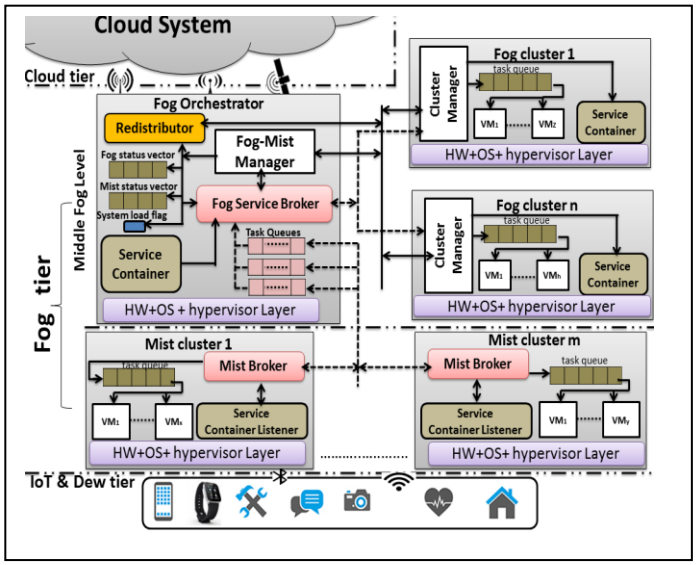
Obviously, any IoT application can be consisted of one or more services. These services create one or more tasks, which will be executed on one or more of resource units (VMs) in cloud, fog, or mist. The discernment in between mist, fog, or cloud in resource allocation is depending on the urgency of the generated tasks by the service. Hence, there are four levels of

task urgency. The first level is the highest time urgency, which required less computation density. This type of tasks can be calculated by the dew computing nodes, which are called ***dew-based-tasks.*** The second level of task urgency is also has a highest time urgency but it required high computation density. This type of tasks will be assigned to one of mist nodes, which is called ***mist-based-tasks***. The third level of task urgency has specific QoS threshold. This type of tasks will be assigned to fog nodes in the middle fog level. Also, this type of tasks is called ***fog-based-tasks***. The last type of tasks, which isn't real-time tasks and required massive amount of resources and huge-volume storage, will be assigned to the cloud nodes. This type is called ***cloud-based-tasks***.

MLITS supports various levels of management mechanisms to deal with the IoT service requirements. Figure 2 depicts the global architecture of the proposed model and functionality of its components. Frist of all, the first type of task urgency is handled by the dew computing node. Consequently, the mist node will receive the remaining types of task urgency (second, third and fourth urgency level). The mist nodes receive these types of tasks by the Service Container Listener (SCL). SCL asks the Mist broker to allocate the required resources for the received tasks. In another word, the SCL directs the tasks to the mist broker.

The mist broker assigns the second type of task urgency (mist based tasks) to its local VMs and forwards the other tasks to the Fog Orchestrator node. The Orchestrator node contains three queues for buffering the received tasks from mist nodes. The Fog Service Broker (FSB) module in the Fog Orchestrator node receives the third and the fourth types of task urgency. Moreover, the Orchestrator node receives the mist base tasks in the case of the mist node doesn't has sufficient resources. Hence, it takes the decision of to process the mist based tasks in one of the middle fog node. Also, Orchestrator node can forward the cloud based tasks to the cloud system. Before FSB sending a task to any of fog cluster it has to check its status. The FSB module uses mist status vector (MSV) and fog status vectors (FSV) to get the least expected waiting time load cluster in the system.

Fig. 2. Multi-Level IoT Tasks scheduling (MLITS) system



The Fog-Mist-Manager (FMM) is responsible for updating the values of MSV and FSV. Each of FSV and MSV are containing set of objects. Each object describes the status of a specific mist or middle fog node in the cluster. Each Fog status object contains variables such as; number of idle VMs, expected waiting time, overload flag, and real flag. The first variable is representing the number of idle VMs of the cluster. The total expected waiting time for each cluster is sent by the cluster manager periodically. Also, this value is expected by FMM after assigning a new task for that cluster. The overload flag is a binary variable. This flag is set by 1, if the expected waiting time for its cluster exceeds the desired quality of service threshold (σ). Finally, if all VMs of a cluster are busy, the cluster manager asks FMM to set real flag by 1. Furthermore, if the load over all the system exceeds σ, the FMM lock the System load flag to direct the fog based tasks to the cloud system.

The following subsection discusses the structure of the mist node. The next subsection describes the components of the Orchestrator node. Section 3.C debates the main component functionality in the fog middle node.

IV.A) Mist node

Service Container Listener (SCL) receives the new added services listener and the up-to-date changes of the services from the Service Container (SC) in fog orchestrator node. Service Container Listener (SCL) directs the new tasks to the Mist Broker. The mist broker determines the type of the service request tasks. If the type of task is mist based, it allocates the task in a local VM, otherwise it send the task to the Orchestrator node. In case of all local VMs in the mist node are busy, the mist broker computes the expected duration time for the new mist based tasks. If the expected duration time doesn't exceed the deadline of the task, mist broker inserts the task in the waiting tasks queue, otherwise it sends the task to the Orchestrator node. In another word, if there are insufficient resources for the mist based tasks, it directs the tasks to the Fog Orchestrator node. Also, mist broker sends the fog base tasks or cloud base to the Orchestrator node. The main steps of the Mist Broker module are shown in the following algorithm.

IV.B) Orchestrator Node

The Fog Service Broker (FSB) receives the service request tasks from the Mist Brokers. FSB allocates the tasks based on its urgency and the available resources, as shown in Fog Service Broker algorithm. In another word, this module is responsible for mapping between the task urgency level and available resources. Moreover, Fog Mist Manager (FMM) is responsible for broadcasting a copy of the Service Container to all fog and mist computing nodes. In other words, FMM should send up-to-date a copy of additional changes in Service Container. The *Service container* contains a copy of the services for IoT application. Also, it contains the requirement description of each application request. In another word, the application developer specifies a set of service requirements; for example the service task urgency level, the hardware requirements, number of resource units (VMs), replication requirement, data movement between services, and

relationship between the services (like remote method invocation).

Mist Broker algorithm

Input
t // service request task

```
1.  If(t.type = mist_based)
2.    IdleVM= getIdelVM()
3.    If(IdleVM ≠ ϕ )
4.          Allocates(t, IdleVM) /*Allocate the task
                                   to idle VM */
5.    Else
6.        t.durationTime = waitingTime() + t.exp ExeTime
          /* compute the expected duration time*/
7.        if ( t.durationTime < t.deadline )
8.            Mist.taskqueue.Enqueue(t)
              /* insert the task in the local task queue*/
9.        Else
10            Orchestrator.mistQueue.Enqueue(t)
              /*Send the t to the mist task queue in
                Orchestrator node */
11        End if
12    End if
13  Elseif((t.type = fog_based))
14        Orchestrator.fogQueue.Enqueue(t)
15  Else
16        Orchestrator.cloudQueue.Enqueue(t)
17  End if
```

FSB receives tasks of second and the third level of urgency from the mist node to choose the best resources for it. FSB allocates the tasks based on resources status information of the system node. The resources status information is written by Fog Mist Manager (FMM) in the Fog status vector and Mist status vector. Each fog node sends up-to-date information about its resources status to FMM periodically or when the cluster status is changed.

Hence, FMM prioritizes the resources allocation for the new tasks based on load and the number of idle VMs in each cluster. FMM periodically get the total execution time in each cluster in the system (mist nodes, and middle fog) from the Cluster Manager. Hence, the cluster load can be defined as total required time for process the assigned tasks. The mean load over all the system (τ) is used by FMM to direct the tasks to each cluster node. A cluster can be classified as high load if it's total waiting time greater than σ. FMM sets the overload_flag = 1 for each high load clusters. Hence, the overload_flag prohibits FSB to assigning more tasks for high load cluster. Moreover, in case of the load of any cluster exceeds σ, FMM calls Redistributor. The Redistributor migrates the tasks from the high to a lowest load cluster, as shown in Redistributor algorithm. In another word, FMM migrates service requests from the high load cluster to the low load clusters. Furthermore, if the system mean load (τ) load exceeds specific threshold (σ), FMM sends the incoming tasks to the cloud system

Fog Service Broker (FSB)algorithm

```
1.  While (|MistTaskQueue|≠null) or (|FogTaskQueue|≠null)or
    (|cloudTaskQueue|≠null)
2.  mistTask = MistTaskQueue.FindMinDeadline()
3.  fogTask  = FogTaskQueue.FindMinDeadline()
4.  cloudTask  = cloudTaskQueue. Dqueue ()
5.  If(mistTask ≠null)
6.      FG = Fog_ status_Vecor.Find_idelVM_fog()  /*find
                                  fog which has idle VMs */
7.      If  FG.idelVM ≠ ϕ       // fog node has idle machine
8.          Send_task( FG, t)   // send the task to the fog node
9.      Else
10          FG = Fog_ status_Vecor. Find_lowLoad_fog()
    /*find the lowest

    load fog */
11          Send_task( FG, t)
12      End if
13  End if
14  If( fogTask≠null)
15      If(System_load_flag= 0)
16          FG = Fog_ status_Vecor. Find_lowLoad_fog()
                /*find the lowest load fog */
17          Send_task( FG, t) // send the task to the fog node
18      Else
19          Send_task( cloud, t) // send the task to the cloud
20      End if
21  End if
22  If( cloudTask ≠ null)       // the task type is cloud base
23      Send_task( cloud, t)
24      End if
25  End while
```

For each cluster *i* , the total load can be computed by the following equation.

$$T_i = (\sum_{j}^{x} wt_j + \sum_{l}^{n_i} Et_l) / \sum_{k}^{n_i} p_k \quad \rightarrow (1)$$

Where:

$\sum_{j}^{x} wt_j$: is total execution time of the waiting task in the waiting list.

$\sum_{l}^{n_i} Et_l$: is total execution time for the interleaved process.

Also, is n_i the number of VMs in the cluster *i*.

$\sum_{l}^{n_i} p_l$: is the total processing power by Million instructions per second (MIPS) for each VM in the cluster *i*.

If the load of a cluster *i* exceeds σ, FMM sets the cluster load-flag =1 in the status vector Fog status vector. Hence the

mean load over all the system (τ) can be computed as follows.

$$\tau = (\sum_{i=1}^{n} T_i) / n \qquad \rightarrow (2)$$

Where n is the number of system clusters.

Fog Mist Manager (FMM) Algorithm

Input:

T_i //the load of each clusters in the system, where $i \in \{1,..,n\}$

```
While (true){
    Wait(q)      // collect up-to-date information after period q .
    For ( ∀c_i ∈ C )      //loop for all clusters
        If ( T_i ≥ σ )
            c_i.overload_flag = 1
            Call Redistributor ()
        End if
    End for
    τ = (Σ_{i=1}^{n} T_i) / n
    If ( τ ≥ σ )     //the system in high load
        System_load_flag=1  /* this flag to direct the load to cloud */
        Call Redistributor()
    End if                //the system in low load
    FSV.sort()     // Status Vector
    MSV.sort()     //sort Mist Status Vector
End while
```

Redistributor Algorithm

```
For ( ∀c_i ∈ C )      // loop over all cluster node
    If ( c_i.overload_flag = 1 )  // if the cluster overloaded
        T_i =getClusterLoad( c_i )   // get the current cluster load
        While( T_i > σ )
            Cmin=findMinLoadCluster()  /* find the lightest cluster load */
            MigratedTask = c_i.FindMaxDeadline()  /* get the latest task */
            If ((MigratedTask+ getClusterLoad(Cmin))< σ )
                Migrate(MigratedTask, Cmin)
            Else
                Migrate(MigratedTask, cloud)
            End if
        End While
    End if
End for
```

Cluster manager algorithm

Input

s // a task

output

T_i // total load

```
If(s.type= mist_based)
    If (Number of idelVM > 0)
        Allocate task s to idle VM
    Else
        Preempt a VM
        Allocate task s to the VM
    End if
Else          // fog based type
    Allocate s task queue
End if
T_i = (Σ_j^x wt_j + Σ_l^{n_i} Et_l) / Σ_k^{n_i} p_k
If ( T_i ≥ σ )
Fog_status_vector.overload_flag = 1
End if
```

IV.C) middle fog nod

The core component of any middle node in the fog system is the Cluster Manager (CM). CM is responsible for monitoring the cluster status and sends it to the Fog Orchestrator. It sends the total expected waiting time periodically. Also, if all cluster VMs are allocated by mist based task, CM asks the Fog Orchestrator to suspend sending real tasks. Hence, the Fog-Mist Manager (FMM) sets the real flag value to one for that cluster object in the Fog status vector. Also, after the finishing one or more real task, CM sends to the Fog Orchestrator to change the real flag to 0. Furthermore, CM preserves QoS for any fog tasks by maintaining σ as an upper bound for fog task waiting time. In other words, the CM request FMM to sets overload_flag=1 when the total waiting time in the cluster exceeds σ . The following table shows the main steps of the Cluster Manager algorithm.

IV. SIMULATION SETUP AND EXPECTED RESULTS

In this section, we will evaluate the feasibility of the MLITS model. To obtain this, the technical details of MLITS will be depicted firstly in Subsection (IV.A). Hence, the assessment of MLITS performance is performed in a two cases. The first case, which is obtained in Subsection (IV.B), measures the system performance using both of soft and real IoT tasks. The second case C inspects the impact of the MLITS model on the real time tasks only, as shown in Subsection (IV.C). In each case, the test is evaluated based on three parameters; the average turnaround time, the average waiting time and the throughput. These parameters are evaluated for the proposed model and all the other competitive scheduling models.

IV.A) Simulation Tool (CloudSim)

To evaluate the proposed model, the fog-mist colony of 100 nodes was built on simulator CloudSim 3.0.2 [27, 28] to execute tasks. This colony is divided into two parts; the first part contains the mist nodes which are the half of total nodes, while the second part contains the middle fog nodes. The fog-mist colony is connected to a cloud system to perform a portion of tasks which doesn't required real time speed. The experiments have been implemented by using Window 7 OS, core i5 2.3 GHz processor and NetBeans IDE 7.2.1.

In this evaluation, the IoT applications are characterized by two types (real and soft). The soft-task size is different from 0.04 to 0.08 million instructions and have 300 MB of incoming and 300 MB of outgoing data. On the other hand, real-task requests are ranged from 0.02, 0.04 million instructions and also have 300 MB of incoming and 300 MB of outgoing data. 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.

In the simulation test, we suppose that each service in the proposed model generates a unique independent task. All test values are repeated for 10 times and the average values are occupied. MLITS model is compared with three models; Min-Min Algorithm, Credit Based Scheduling (CBS) Algorithm [26], Improved Priority based Job Scheduling Algorithm [27], and Earliest Feasible Deadline First (EFDF) [28].

IV.B) performance measurement for All types of tasks

This section contains three experiments. The first experiment measures the growing of turnaround time as increasing the number of service requests. The second test assesses the waiting time of the system. Finally, the last test compares the models throughput.

Turnaround time performance test: the turnaround time for a task is defined as the time taken to complete the task. The performance comparison, which is based on the Turnaround time parameter, is shown in Figure 3.A. All of the tests are completed using several workloads, which starts from 1000 up to 10,000 tasks. The ration of the real time tasks is 20% from all of the integrated workload in each test. Obviously, the Mini-Mini curve is increasing rapidly as the load increased. The high increasing of the Mini-Mini turnaround time is caused by high priority of the short tasks. In another word, Mini-Mini algorithm increases the waiting time of the massive processing tasks. The CBS model is the closest curve to the proposed model since it uses both of the priority task and size of the tasks. Unfortunately, CBS and EFDF are suffering from the unbalanced load. On the other hand, MLITS model gives the best performance since it uses different level of priority and load balancing. MLITS overcome the other model by allocating the task according to its level of urgency with considering the load balance constraints.

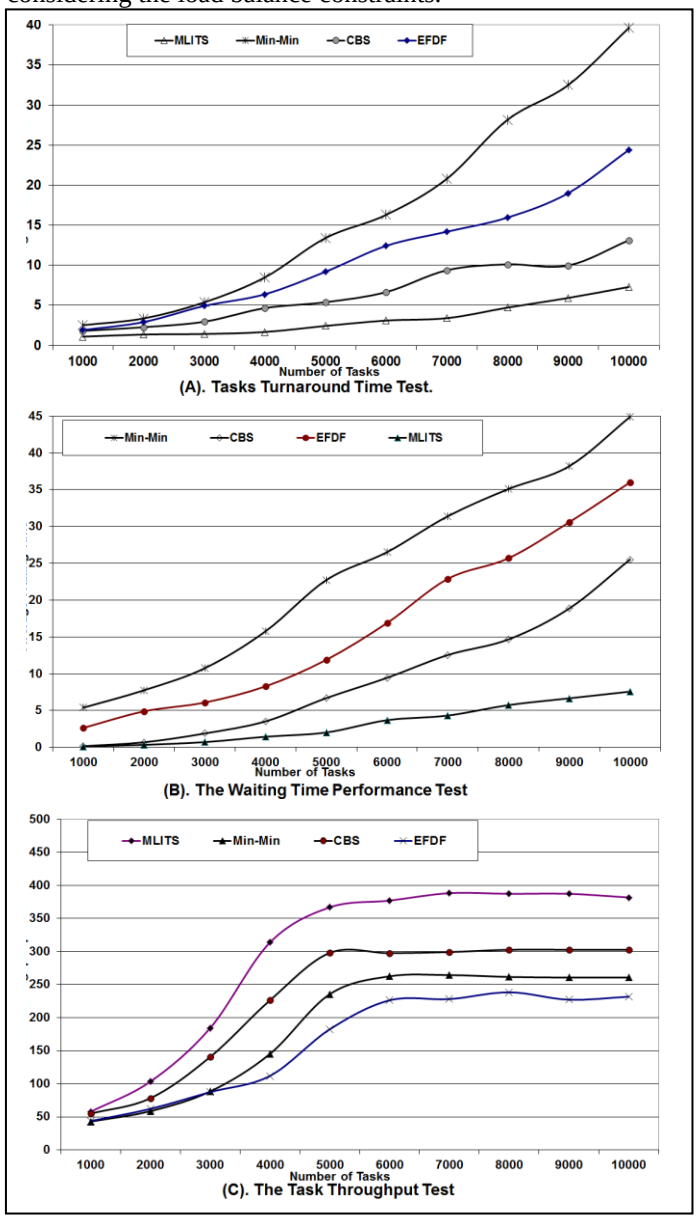


Fig. 3. Performance Comparision using Soft & Real-Time tasks

The waiting time performance test: The second experiment in this test is waiting time performance test. In this experiment, the performance evaluation is based on the average of the waiting time parameter. This experiment is done based on the same system load of the previous experiment. The waiting time curves of the compared

algorithms are obtained in Figure 3.B. The Mini-Min gives the highest waiting time inasmuch it allocates the shortst task will allocated in the fastest resources. The reason for the high increase in Mini-Min waiting time is due to allocating the fast resources to the short tasks. Therefore, the long tasks will be allocated in the slow resources or it will be starved. The reason for the of the waiting time for Mini-Min is that the fast resources will be allocated to the short tasks will the long tasks will have slow resources or starving .

Also, at the low load of services requests the CBS is much closer to the proposed model. Moreover, the waiting time of CBS and EFDF curves are increased dramatically. This is caused by long tasks are starving.

The throughput performance test: The throughput is calculated as the number of the finished tasks per unite time. Figure 3.C obtains a throughput comparison between the competitive models. Similarly, this test is accomplished on the same workload of the previous experiments. EFDF model gives the worst throughput, since its scheduling model is only based on the task deadline. Also, at the low load the Min-Min is so closed to the EFDF model. Furthermore, the differences between the curves are low in the light load of the system, but as increasing the system load the performance of the MLITS is overpass.

IV.C) Real Task performance measurement

This test is carrying out to evaluate the influence of the proposed system on the real time tasks. This test is carried out to evaluate the influence of the proposed system on the real time tasks. Tests are reiterated for 10 times. In each experiment the ratio of the real time requests is 20% from the inserted workload.

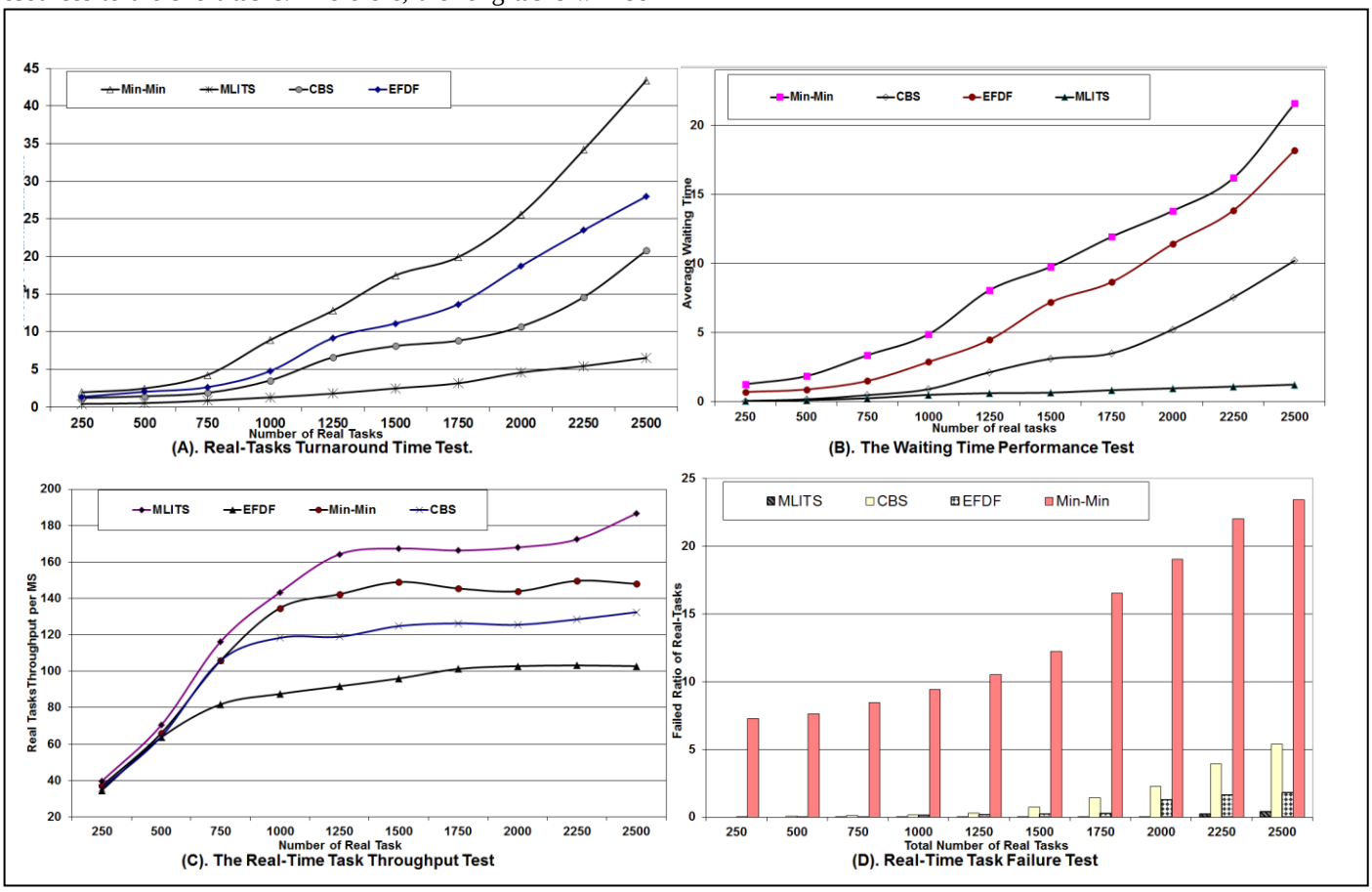


Fig. 4. Real-Tasks Turnaround Time Test.

This section exhibits the influence of the compared models on the real-time services requests. The following experiments are done using the same test parameters as previous tests.

Turnaround time performance test: The first experiment measures the average turnaround time, as shown in Figure 4.A. The worst results are obtained by the curves that represent the Min-Min, CBS and the EFDF respectively. However, the main shortages of the previous models are the disability to load balancing and serving the real time tasks according to their urgency level. I.e. the task urgency type determines the type of the allocated VM (mist, fog, or cloud). Min-Min has the worst performance since it doesn't distinguish between the real or soft tasks. In the other hand, CBS and EFDF distinguish between the real and the soft tasks but it can't distinguish the types of real tasks.

The waiting time performance test: The averages of waiting time curves that expose the impact of the MLITS model on the finishing of the real tasks are shown in Figure 4.B. In this figure, the MLITS has the least waiting time. In point of fact, the prosed model gives the highest priority to the

real tasks. Also, the load balancing strategy and the classification of the real time tasks are contributing rapid the process of resource allocations. However, the CBS is the closest curve to the MLITS through overall compared model. Unfortunately, the increasing of the number of tasks in CSS model is increasing its intermediate layer delay. Therefore, the deadline for the real time service may be broken for the other methods.

The throughput performance test: The throughputs curves of the compared models are obtained in Figure 4.C. Obviously, the best throughput is completed by MLITS model. The throughput results oscillate between 97% and 100% for all the inputs of the variant loads for the real tasks. Obviously, MLITS has the highest throughput. The high throughput of MLITS is caused by load balancing among the different level of mist-fog-cloud architecture. Since CBS give the short task to the speediest resources, therefore the closest curve to the MLITS is CBS.

To judge about the suitability of the algorithm for real time services, the task failure should be concerned. Figure 4.D measures the ratio of the failed real time tasks for each compared model. The ratio of the failed tasks for the MLITS model can be neglected if compared with the other models. CBS and EFDF models have an acceptable performance in low load of the real time tasks. But, CBS is concerned on the task size with the task priority in resources allocation. Due to resources allocation strategy in CBS depends two factors, the task size in additional to the priority, which it causes increasing in failure ratio in case of high load system. In another hand, EFDF scheduling method reduces the failure rate by concerning only on the deadline. But, this strategy is failed without considering the type of the resources and the service requirement.

V. CONCLUSION AND FUTURE WORK

In this paper, MLITS model is designed for integrate the mist with fog and cloud computing environment to enhance the data collection and processing in IoT environment. The proposed model is responsible for handling soft and real time tasks with different type of urgency in Mist-Fog-Cloud colony. The MLITS is an orchestration model that fit the rapid increasing scale of IoT deployments. the proposed model maintains the load balance among the extreme edge node of fog and the middle fog node. Moreover, MLITS classify the tasks by the urgency and the computation density. This model is designed to handle the real time tasks in additional to maintain QoS for the soft-tasks. The experiments exhibit that the proposed model outperforms the compared models. In future work, this model will be developed to manage the heterogeneous resources.

REFERENCES

[1] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer and Shahid Khan, ”Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges,” in Proceedings of Frontiers of Information Technology (FIT), 2012, pp. 257-260

[2] J. Zheng, D. Simplot-Ryl, C. Bisdikian, and H. Mouftah, “The Internet of Things,” in IEEE Communications Magazine, Volume:49 , Issue: 11, 2011, pp:30-31.

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

[4] Alessio Botta et al. ,“On the Integraton of Cloud Ccomputing and Internet of Things ”, International Conference on Future Internet of Things and Cloud, 2014, pp. 9-21

[5] Gomes, M. M., Righi, R. d. R., da Costa, C. A. Future directions for providing better iot infrastructure. In: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication. UbiComp ’14 Adjunct. 2014, pp. 51–54.

[6] Alhakbani, N., Hassan, M. M., Hossain, M. A., Alnuem, M. A frame-work of adaptive interaction support in cloud-based internet of things (iot) envi-ronment. In: Internet and Distributed Computing Systems. Springer, 2014, pp. 136–146.

[7] Ballon, P., Glidden, J., Kranas, P., Menychtas, A., Ruston, S., Van Der Graaf, S. Is there a need for a cloud platform for european smart cities? In: eChallenges e-2011 Conference Proceedings, IIMC International Information Management Corporation. 2011, pp. 1-7

[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, Springer. 2009, pp. 70–80.

[9] Sarkar, S., Misra, S., “Theoretical modelling of fog computing: a green com-puting paradigm to support iot applications”, IET Networks 5(2), 2016, pp. 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., 48, (7), 2015, pp. 37–45

[14] Bonomi, F., Milito, R., Zhu, J., et al.: ‘Fog computing and its role in the in-ternet of things’. Proc. of the First Edition of the MCC Workshop on Mobile Cloud Computing (ACM), Helsinki, Finland, August 2012, pp. 13–16

[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, 2014, 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), 2015, pp. 37–45.

[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, pp 4049-4062.

[20] Fei Tao, Meng Zhang and A.Y.C. Nee. "Chapter 8 - Digital Twin and Cloud, Fog, Edge Computing" Academic Press, 2019, pp. 171-181

[21] Subhadeep Sarkar and Sudip Misra. Theoretical Modelling of Fog Compu-ting: agreen Computing Paradigm to Support IoT Applications.IET Networks, 5(2), 2016, pp. 23–29,

[22] Giang, N.K.; Blackstock, M.; Lea, R.; Leung, V.C. Developing IoT applica-tions in the Fog: A distributed data flow approach. In

Proceedings of the 2015 IEEE 5th International Conference on the Internet of Things(IOT), Seoul, Korea. 2015; pp. 155–162

[23] Y. Liu, J. E. Fieldsend, and G. Min, ‘‘A framework of fog computing: Archi-tecture, challenges, and optimization,’’ IEEE Access, vol. 5, 2017, pp. 25445–25454.

[24] Barik, Rabindra Kumar et al. “Mist Data: Leveraging Mist Computing for Secure and Scalable Architecture for Smart and Connected Health.” 2018, pp. 647 – 653.

[25] Soheil Anousha and Mahmoud Ahmadi, "An Improved MinMin Task Scheduling Algorithm in Grid Computing," Springer-Verlag Berlin Heidelberg, 2013, pp. 103-113.

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

[27] Patel, Swachil J. and Upendra R. Bhoi. “Improved Priority Based Job Sched-uling Algorithm in Cloud Computing Using Iterative Method.” 2014 Fourth Inter-national Conference on Advances in Computing and Communications, 2014, pp. 199-202.

[28] Jagbeer Singh, Bichitrananda Patra, Satyendra Prasad Singh, "An Algorithm to Reduce the Time Complexity of Earliest Deadline First Scheduling Algorithm in Real-Time System" (IJACSA) International Journal of Advanced Computer Science and Applications, February 2011, pp.31-37.

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.