Informatics Bulletin, Faculty of Computers and Artificial Intelligence, Helwan University
Published Online ***Vol 4 Issue 1, January 2022***
(https://fcihib.journals.ekb.eg)

RTSIF: Real-Time Scheduling on IoT-Fog Framework for Managing Applications in Smart Cities

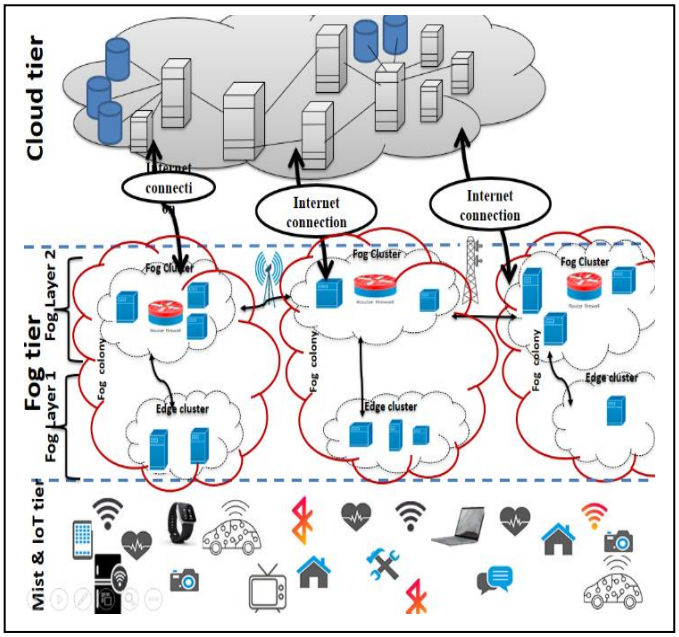
Hosam E. Refaat[1], and Mohamed A. Mead[2]
[1]Dept. of Information System
hosam.refaat@ci.suez.edu.eg
[2]Dept. of Computer Science
mohamedmead@ci.suez.edu.eg
Faculty of Computers and Informatics
Suez Canal University, Egypt.

The existence of smart cities and the diversity of their applications has now become a necessity that helps in reducing costs and managing available resources efficiently. IoT applications help to expand and support smart cities, as it manages and monitors devices and acquire rapidly a collection of big data to benefit from it. Smart cities applications vary in terms of the types of data generated, the need for quickly data analysis, response speed, or the need for massive data storage. Fog and mist computing models provide solutions for all these requirements, as they can analyze and store big data and interact with IoT devices quickly and smoothly. In this research a fault tolerant model will be proposed to support and manage applications in smart cities. This model relies on three tiers of computing resources (Mist, Fog, and Cloud). The proposed model distributes processing tasks to the edge to reduce data latency and support real-time applications in smart cities. In addition to, the IoT tasks are classified in this research based on their deadline and the urgency of their execution. Also, the performance of the proposed model is measured and compared with three scheduling models, namely; Min-Min, Credit-Based-Scheduling (CBS) and Earliest-Feasible-Deadline-First (EFDF). Through comprehensive simulations, the test measurements obtain improvement in the performance metrics.

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

I. INTRODUCTION

With the concept of the Internet of Things, a new era of connecting devices over the Internet to collect, send and receive data was raised. The architecture of IoT for receiving, storing and analyzing data is varied, as all of these operations can be performed on a cloud system [1, 2]. However, because IoT applications are mostly latency-sensitive applications, the computing that takes place in cloud computing services is insufficient for these applications and leads to the failure of these applications. The most successful solution is to add computing and storage bits to be close to IoT devices. The concept of fog computing was introduced to solve a latency problem, but at the same time it added a burden on designing systems to use the available resources [3]. Hence, the concept of Mist computing was introduced to bring computing closer to IoT devices, which increases the response speed of IoT applications [4,5]. Figure 1 illustrates these computing hierarchy that must be used to increase the efficiency of IoT Applications. One or more task can be generated in each request to an IoT Application or IoT sensor triggering action. These tasks should be allocated in computation resources. The most of the previous resources allocation model are suffering from the un-balancing load. Also, some of these scheduling model is based on the first deadline



first, hence allocate the tasks in the idle machine without considering the characteristics of the machine and the type of the IoT application.

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

In this paper, IoT tasks will be divided into two main parts, the first being non real-time tasks and the second being real-time tasks [6, 7]. Non real-time tasks such as information mining and storing data do not require high response speed, as they also require large storage space in addition to high computing power. Therefore, it is possible to rely on the cloud layer for executing this type of IoT tasks. On the contrary, real-time tasks need a high response speed, and this type of task can be divided into hard real-time, soft real time and firm real time based on their urgency [8]. Such example of this type of tasks is the Intelligent Transportation such as Collect data from roadside sensors and cameras.

Soft real-time tasks have a deadline without absolute values, but at the same time there is no system failure or change in results when a deadline is exceeded. An example of this type is the face recognition task [9]. On the other hand, hard real-time tasks adhere strictly to a deadline, as exceeding a deadline leads to major problems, accidents and system failures such as the tasks of self-driving cars. Finally, Firm Real-time Tasks are similar to Hard Real-time Tasks, but a deadline can be allowed to be exceeded with little probability. An example of this type of task is video conferencing, where sending data needs a deadline with a relatively small possibility to bypass this deadline, which if it happens does not lead to huge problems or failure for the system as a whole.

This paper proposes a new fault tolerance model called "RTSIF". This model has been proposed over Mist-Fog-Cloud colony for serving the real time applications. The RTSIF has three objectives. The first objective is to serve the Real time tasks based on its type (hard, soft and firm). The second objective is aimed to provide persistent services by building providing fault-tolerance system. The third objective is maintaining the workload balanced between the fog nodes. Moreover, RTSIF maintains the resources of the mist node able to serve the hard real-time tasks.

II. RELATED WORK

In Edge computing, data processing is done near the edge where data is generated from IoT devices [10]. In this case, various applications in smart cities, especially applications that require real-time data for their operations, benefit from the proximity of storage and processing places. In [10], an Internet-scale repository (GigaSight) of crowd-sourced video content is proposed. Where it relied on small data centers close to IoT devices that send data, these centers can receive and analyze data in real time based on the concept of fog computing.

In [11], the authors proposed a platform based on fog computing to analyze the big data generated by IoT devices. In this research, IoT data captured from smart homes was analyzed by fog computing nodes that help in in-time decision making. In [12], the researchers presented a solution to the parking problem based on fog computing. Data is collected automatically (number of vacant spaces and number of vehicles that want to park) through fog nodes to give drivers real-time information about the nearest vacant parking spot. On the same approach, the researchers presented in [13] a system to provide updated information in real time about the arrival and departure times of public transport buses through fog nodes.

In [14], a real-time traffic management system based on the cloudlet layer was introduced to reduce response time. Information from vehicles and sensors that represent road fog nodes are collected and analyzed on the cloudlet layer and then sent to the cloud layer. In [15], fog and mist nodes were used to bring computing closer to the edge of the Internet of Things architecture, with decentralization support, which increases the system's response speed in decision-making.

In [16], the Internet of Healthcare Things (IoHT) framework is designed to lower healthcare costs while delivering high quality and reliable services. This framework relied on mist, fog and cloud layers to allocate and use resources efficiently while processing data in real time. In this framework, sensitive data is given priority in its transfer and processing, with policies for data transfer based on the data source.

Given the resource consumption of IoT devices and the slow response time of IoT applications, in [17] a framework based on the concept of mist (a cloud near the earth's surface with lesser density than fog) is proposed. This framework makes computation and storage close to the edge of networks, in order to conserve resources and speed up response time for any IoT application.

Zoltán [18] introduces a framework that based on three layers (IoT, fog, cloud). This model introduces different viewpoints to serve variant disciplines. Unfortunately, this model doesn't cover the real time services.

III. PROPOSED MODEL

The proposed model (RTSIF) contains three tiers of computing resources (Mist, Fog, and cloud). These three tiers and their interactions with each other in RTSIF model is represented in figure 2. The first tier (front-end) of the model consists of mist devices through which users can submit their requests over different types of communication. This front-end is also called Mist tier. Briefly, the Mist is a solution for hard-real time IoT applications, such as firefighting system. This tier is presenting as interface for serving the IoT devices and user requests. The IoT devices is connected to the mist nodes via low-latency network connections to recompense IoT strict constraints on their resource such as CPU, memory, and, when run, a very complex application. Hence, the mist tier for this kind of applications is responsible for interact with the world. The main scenario for real time application based on two actions, the first actions is collecting the data from the environment which is done by the IoT sensors. In other hand, the second action is done by the mist nodes that controlling the actors which have an effect on environment. Moreover, this tier improves reply time for other type of services (non-real-time, soft-real-time and firm-real-time services) by choosing the most suitable resources to each type of service requests.

Unfortunately, the front-end tier suffers from limited memory and computational resources, that forces us to send the complex

tasks to the second tier which is the near-end fog nodes. The cloud system is representing the third tier. This tier is dedicated for the non-real time services which require massive computations resources.

The Mist tier is specified for the hard-real time applications, which generated tasks can be classified as "Mist-based" tasks. Also, the Mist-based class of tasks can include lite firm-real time applications. The lite firm-real time application required lite processing power and storage, with a specific QoS. The lite firm-real time tasks are allocated in the mist nodes, if the sufficient resource is available, as defined by Equation 1. Generally, this type of tasks generated by fault-tolerance applications, which require persistent services, and computation resources with a negligible delay. In another word, to have persistent services this tasks should have a hot replica.

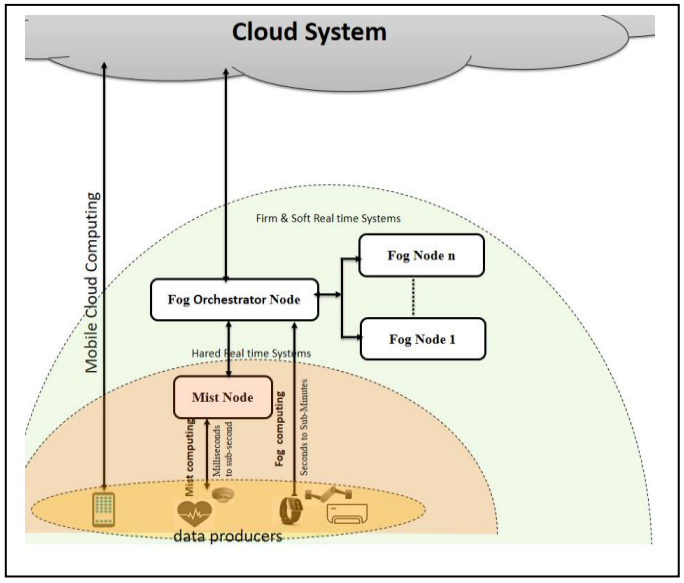


Fig. 2. General Overview of RTSIF system.

$$\tau_{Mist} = \begin{cases} \tau_{h_i} & , \quad \forall i \\ \tau_{f_j} & , \tau_{f_j}.deadline < \alpha \end{cases} \rightarrow \quad (1)$$

Where, τ_{h_i} and τ_{f_j} is the hard-real time and firm-real time tasks respectively. Also, α is a threshold used to determine if the task can be allocated in mist or fog node. In another word, if the deadline of a task is less a threshold(α), the task should be allocated in the mist tier. Also, this threshold is summation of the duration time of the task in fog node and the communication overhead, as defined by Equation 2. The expected duration time of a task in a fog node is defined by the Worst Case Execution Time (WCET).

α = *expected fog duration time + communication cost* → (2)

On another hand, the firm-real time, and soft-real time application will be directed to the second tier of the system, which is called "Fog tier". The Fog tier resources are managed by the orchestrator node. The orchestrator node can receive all types of the tasks form the mist nodes. Hence, it is responsible for allocating these tasks based on the task urgency in the available resources with minimum delay. Also, the orchestrator node is responsible increasing the availability of the mist layer by providing hot replication for the persistent services. The last type of task is non-real time that require massive computations, which is called "Cloud-based". As missioned before, the proposed model is consisted of three layers. The details of RTSIF model layer architecture is shown in Figure 3. In the next subsection the structure of the mist tier and the fog tier will be discussed.

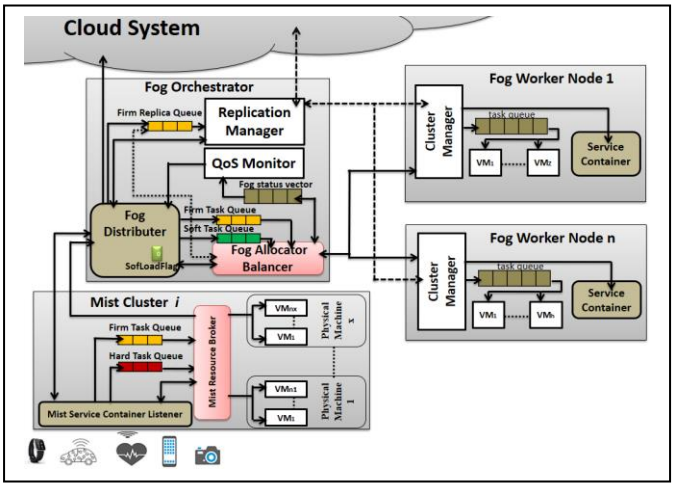


Fig. 3. RTSIF System Architecture.

A. Mist Tier

Each node in the mist tier consists of two main components, namely; Mist Service Container Listener (MSCL) and Mist Resource Broker (MRB). MSCL is responsible for receiving the IoT service requests. In another hand, Mist Resource Broker is allocating the tasks in the local resources or direct it to the next tier. The following subsections discuss the main functionality of the mist node components.

A.1. Mist Service Container Listener (MSCL)

MSCL is responsible for receiving services request and data from the user or IoT. The services requests are represented by tasks which will be assigned to resources in mist, fog or cloud system. MSCL classifies the service request, as shown in MSCL algorithm (Algorithm 1). The MSCL algorithm insert all of the hard real time tasks in the hard task queue, which will be allocated in the local mist resources by the MRB. Moreover, MSCL classified the Firm-real-time tasks according to Equation 1 into Mist-base or Fog-Base tasks. In another word, if the task deadline is less than the cost of accomplishing the task in fog node in additional to the communication overhead (as defined in Equation 2), the task is allocated in the mist resources. If the service request is classified as Mist-Based task, service request is directed to the Mist Resource Broker (MRB). On the other hand, the other types of requests will also be redirected to the Fog Orchestrator Node (FON).

Algorithm 1: MSCL Algorithm

Input

t // new service request

Step 1: If (*t.type* = "*hard"*)
Step 2: Mist.hardQueue.Enqueue(t)
/* insert the task in the hard realtime queue*/
Step 3: Elseif(*t.type* = "*Firm"*)
Step 4: If(*t.deadline*< α)
Step 5: Mist.FirmQueue.Enqueue(t)
Step 6: Else
Step 7: Orchestrator.fogQueue.Enqueue(t)
/*Send firm task to Orchestrator node queue */
Step 8: End if
Step 9: Else
Step 10: Orchestrator.fogQueue.Enqueue(t)
/*Send firm task to Orchestrator node queue */
Step 11: end if

A.2. Mist Resource Broker (MRB)

MRB receives tasks via two types of queues, namely; Hard-task and firm-real time queue. The hard-real-time task is allocated by the algorithm 2, which provide these tasks the highest priority in resource allocation. The second queue is the Firm-task queue. The Firm-task queue contains the firm-real time tasks, which have deadline is less than α. MRB provides the hard-real time tasks the first priority in resource allocation by the following actions. If all mist cluster VMs are busy, MRB is preempting the heights deadline of the allocating firm-real-time task. Most of the hard real time services required to be fault-tolerant services, these services are called persistent services. To speed up allocating the VM for the hard real time task, the preemption operation must be reduced. Hence, the allocation of Firm tasks is done under certain condition of firm resource occupancy (Ω), as shown in Algorithm 3. Where Ω is threshold determine the maximum number of concurrent VMs can be allocated to firm tasks. Hence, the number of allocated VMs for the Firm real time tasks must not exceed Ω. The firm resource occupancy (Ω) must be rectify periodically to have best resource usage. If there are ω_{i-1} idle VMs in period number $i-1$, Ω is increased by this number in the next period (i). On the other hand, if there are no idle VMs ($\omega_{i-1}=0$) , Ω is decreased by p_{i-1} . Where, p_{i-1} is the set of concurrent preempting operations. The firm resource occupancy (Ω) is corrected periodically based on the following equations.

$$\Omega_i = \begin{cases} \Omega_{i-1} + \omega_{i-1} & , \ \omega_{i-1} > 0 \\ \Omega_{i-1} - Max(p_{i-1}) & , \ \omega_{i-1} = 0 \end{cases} \rightarrow \quad (3)$$

The following algorithms discuss process for allocating the hard and firm real time tasks.

Algorithm 2: Hard Real Time Task Allocating algorithm

Step 1: while(HardTaskQueue ≠ φ)
Step 2: τ = HardTaskQueue.equeue() // take hard task from the queue
Step 3: V = find an idle VM
Step 4: if V = φ // there is no idle VM
Step 5: V = max$_{\forall i}$ ($f_i.deadline$) /* where f_i is firm task in running state */
Step 6: preempt firm task that in V
Step 7: end if
Step 8: allocate τ in V
Step 9: If τ is fault-tolerance
Step 10: $\tau' = \tau$ // τ' is a replication of τ
Step 11: V' = find an idle VM : V'$\in M', V \in M, M \neq M'$
//M ,M' is physical mist node.
Step 12: if V' = φ // there is no idle VM
Step 13: V' = max$_{\forall i}$ ($f_i.deadline$): $f_i \in M'$, $\tau \in M, M \neq M'$
/* τ and its replica τ' will not be allocated in the same physical machine*/
Step 14: preempt firm task that in V'
Step 15: p_{i-1} ++//increase preempt operations in the logs
Step 16: end if
Step 17: end if
Step 18: allocate τ' in V'
Step 19: end while

Algorithm 3: Firm Real Time Task Allocating algorithm

Step 1: while(FirmTaskQueue ≠ φ)
Step 2: V = find an idle VM
Step 3: if V ≠ φ // there is no idle VM
Step 4: if (++FirmVM < Ω)
/* where FirmVM is the number of VM that allocated by firm tasks */
Step 5: f = FirmTaskQueue.equeue() /* take Firm task from the queue */
Step 6: allocate f in V
Step 7: else
Step 8: wait(δ) // wait a period of time δ
Step 9: end if
Step 10: If f is fault-tolerance
Step 11: $f' = f$ // f' is a replication of f
Step 12: if (++FirmVM < Ω)
Step 13: allocate f' in V' // where $V' \neq V$
Step 14: end if
Step 15: end while

B. Fog Orchestrator Node (FON)

As discussed before, the Mist Resource Brokers and The Mist Service Container Listener (MSCL) forward the Soft and Non-Real time tasks to the fog orchestration node. Additionally, the Firm tasks can be forward to the orchestration node if mist node is overloaded. The Fog Distributer receives these types of tasks. Hence, Fog Distributer inserts the soft and firm real time tasks its associated queue. The Fog Distributer contains a copy of all IoT services and encompasses the requirements of each service request. In another word, Fog Distributer has details of each service requirements; such as the persistency of the service, the VM specifications, number of VMs, priority level, data flow between services, and dependency between the services (like remote method invocation). Also, as increasing the work load over the fog nodes, as the FON gives the highest priority to the Firm tasks. Fog Distributer sends the CloudBase (Non-Real time tasks) tasks to the cloud system. The following subsection discusses the main modules of orchestration node.

B.1. Fog Distributer

Fog Distributer receives tasks of service requests form two modules exists in the mist nodes, namely; Mist Service Container Listener (MSCL) and Mist Resource Broker (MSB). Fog Distributer inserts each task in to appropriate queue. When Fog Distributer receives a "***SofLoadFlag***" from the QoS Monitor, it directs the any new soft tasks to the cloud system, as shown in the following algorithm.

Algorithm 4: Fog Distributer Function

Step 1: if t.type= "firm" // chick the task type
Step 2: FirmQueue.Enqueue(t)
Step 3: else task.type= "soft"
Step 4: if $SofLoadFlag$= 0
Step 5: SoftQueue.Enqueue(t)
Step 6: else // Replica of Firm task
Step 7: cloud(t) //sent *t* to the cloud
Step 8: end if
Step 9: else // Replica of Firm task
Step 10: ReplicaQueue.Enqueue(t)
Step 11: end if

B.2. Fog Allocator Balancer (FAB)

Fog Allocator Balancer (FAB) distributes the tasks in different types of waiting queues (firm task, firm replica and soft task), as shown in Figure 3. FAB gives the firm tasks and its replicas tasks higher priority than soft task. Moreover, the tasks distribution process should maintain the load balanced among the fog nodes, as shown by the following equation.

$$w_1 \cong w_2 \cong \cdots \cong w_n \quad : \quad \left|\max_{\forall i}(w_i) - \min_{\forall i}(w_i)\right| \leq \delta \quad \rightarrow \quad (5)$$

$$\delta = \max_{\forall j}(t_j.expExe) \quad \rightarrow \quad (6)$$

Where w_i is the expected waiting time if the task will be allocated in fog node (i). Also, ($t_j.expExe$) is the expected execution time for the task t_j. Equation 5 grantees a balanced load between the fog nodes. Equation 5 means that the maximum load different among all fog nodes must not exceeds δ. Equation 6 defines δ, which represent the longest task execution time. These equations mean that the maximum difference in the waiting time between all nodes must not exceed the execution time of firm task. Hence, FAB maintains the workload balanced among the fog nodes by enforcing these rules.

Furthermore, it should be considered that a fog worker node is considered as overloaded if the waiting time exceed the cost of sending task to the cloud system. Hence, that the maximum of the waiting time over all fog nodes must not exceed η, which represent the turnaround time in the cloud system as shown in the following equation.

$$\max_{\forall i}(w_i) < \eta \quad \rightarrow \quad (7)$$

If the condition of equation 7 is exceed, the SofLoadFlag is set by one to pause the distribution soft tasks on the fog tier. The FAB share the resources based on the QoS ratio which determine the ratio between the soft and firm tasks in fog node, which is denoted by μ. Hence, the expected waiting time for a task in a fog node can be computed by the following equation.

$$w_i = (\upsilon_f(1-\mu) + \upsilon_s.\mu)N_i \quad \rightarrow \quad (8)$$

Where υ_s *and* υ_f is the average of the execution time for soft and firm task respectively. Also. N_i is the total number of tasks in fog node *i*.

Algorithm 5: Fog Allocator Balancer (FAB) Algorithm

Step 1: while(FirmTaskQueue ≠ φ) or (SoftTaskQueue ≠ φ)
Step 2: r = FirmTaskQueue.size*(1-μ) // number of firm
Step 3: f_h = find the minimum fog load
Step 4: f_l = find the maximum fog load
Step 5: for all ($f_i \in F : f_i.load <$ η)
// loop over light load fog nodes e
Step 6: if($f_h - f_l$)< δ
Step 7: distribute firm (1-μ) task
Step 8: distribute soft (μ) task
Step 9: end For
Step 10: end while

distribute firm (1-μ) task

Step 1: for all fogNode.Load < η // the load of the fog node
Step 2: if($f_h - f_l$)< δ
Step 3: $d_f = \frac{f_h - f_l}{\upsilon_f}$
// distance between minimum and maximum
Step 4: $d_i = \frac{f_h - f_i}{\upsilon_f}$
// distance between minimum and maximum
Step 5: $d_i.sent$(FirmTaskQueue.dequeue(d_i))
// send d tasks to f_i node

B.3. Replication Manager

The Replication Manager receives the tasks replications via Firm Replication Queue. Consequently, it allocates each replica task in different cluster form the original allocated task cluster. In another word, if task τ' is replication for τ, then τ' is allocated in V' and τ is allocated in V. Where, V and V' are two virtual machine not hosted in the same physical machine. The following algorithm demonstrates the main steps in allocating replica task τ' which is a replica for task τ.

Algorithm 6: Replication Manager algorithm

Step 1: while(FirmRepQueue ≠ φ)
Step 2: τ = FirmRepQueue.equeue() // take replica task from the queue
Step 3: if $SofLoadFlag$= 0
Step 4: cloud. Send(τ') //send τ' to cloud
Step 5: else
Step 6: V' = find Min load VM : V'∈ M', V ∈ M , $M \neq M'$
//M ,M' is physical mist node.
Step 7: end if

B.4. QoS Monitor

The QoS Monitor is responsible for observer all fog nodes in the system. Equation 7 defines upper bound of the waiting list in each fog worker node in the model. In case of a node load exceed the cost of the η, the other node of the system has load closed to η, as defined by Equation 5. At this case the system forward the new received message to the cloud system until period of time δ, which is defined by Equation 6. After δ period of time the system all fog worker node can accept at least one additional task. Hence, the SofLoadFlag will be reset by zero.

IV. SIMULATION SETUP AND EXPECTED RESULTS

This section measures the Achievability of the RTSIF model. First of all, the technical details of the test environment parameters are described in Subsection IV.A. The evaluation of RTSIF performance is achieved in a two dimensions. The first dimension is concerning in the evaluation of the RTSIF system performance using all type of tasks, which is shown in Subsection (IV.B). The performance measurement is based on three parameters; the average of turnaround time, the average of waiting time and the throughput. Finally, the second dimension evaluates the persistency of the model for the real-time services by evaluating the number of failed tasks in the compared algorithms, which is shown in Subsection (IV.C).

IV.A) Simulation Tool (CloudSim)

WorkflowSim [19] is a simulation program designed as a development of CloudSim [20], which was used to evaluate the model proposed in this paper. In order to assess the test results, the WorkflowSim is used to simulate the compared scheduling models. The WorkflowSim is an open source workflow simulator which is an extension of the CloudSim [20]. The simulation evaluation is done by using the homogeneous characteristics in the Fog and Mist nodes. Fog and Mist nodes has the same characteristics of the VMs in the Amazon EC2. Hence, each task is executed on a T2. Micro instance of Amazon EC2, which is available for free. Fog-mist colony is constructed of 100 fog nodes, in addition to 50 mist nodes. The proposed RTSIF model was compared to three other models, namely Min-Min Algorithm, Credit Based Scheduling (CBS) Algorithm [21], and Earliest Feasible Deadline First (EFDF) [22]. Finally, to perform IoT tasks on these nodes, Windows 10 operating system, Core i5 processor at 2.3 GHz and NetBeans IDE 7.2.1 were used.

IV.B) performance measurement

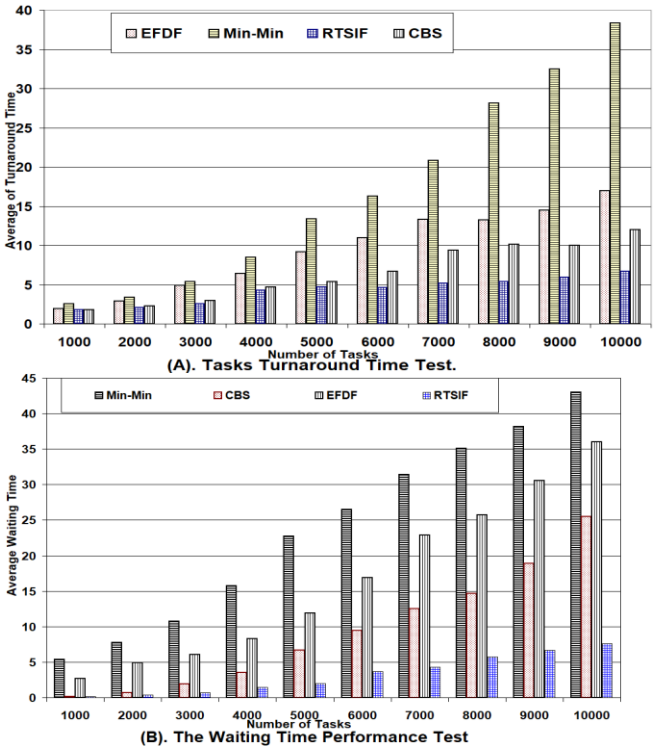
This test composed of three experiments. The first experiment assesses the growing of turnaround time as increasing the number of tasks. The effect of alteration of workload on the waiting time is measured in the second experiment. Finally, the third experiment measures the throughput of the compared models.

Turnaround time performance test: the turnaround time for a task is the time taken to fulfill a service request. Figure 4.A shows the performance measurement based on the turnaround time parameter. The performance test is done using variant workload starts from one thousand tasks up to ten thousand tasks. The ratio of all types of real time tasks are 30% over all integrated service requests. Noticeably, the Mini-Mini bars are increased rapidly as the workload is increased. The poorest performance of the Mini-Mini turnaround time is caused priority of the short tasks. In another word, the high priority that given to small tasks size causes starving in the long tasks.

For CBS, it approximates the performance of the proposed model RTSIF in the case of a small workload of the system. The CBS model allocates tasks based on their priority, while the RTSIF model allocates tasks based on several levels of priority. RTSIF also has the advantage of focusing on urgency of task taking into account load balance constraints

***The waiting time performance test*:** The second experiment measures the waiting time in each model, as shown in Figure 4.B. In this test the waiting time assessment is done by taking the average of the waiting time. The waiting time values of the compared models are obtained in Figure 4.B. The Mini-Min has the worst waiting time with reason that the shortest task will allocated first. On another hand, the long tasks will be postponed or starved until the short tasks is served. Also, the proposed model has performance enhancement since it allocates each task based on its type and the available resources under the load balancing conditions.

***The throughput performance test*:** the experiment, that shown in Figure 4.C, achieves the throughput comparison between the competitive models. The RTSIF has the heights throughput is attributable to the load balancing in each tier of the model. Also, the resources are chosen for each task based on the resource availability and the urgency level of the tasks. The litter difference between the competitive models at the low workload be ascribed to the abundance of resources.



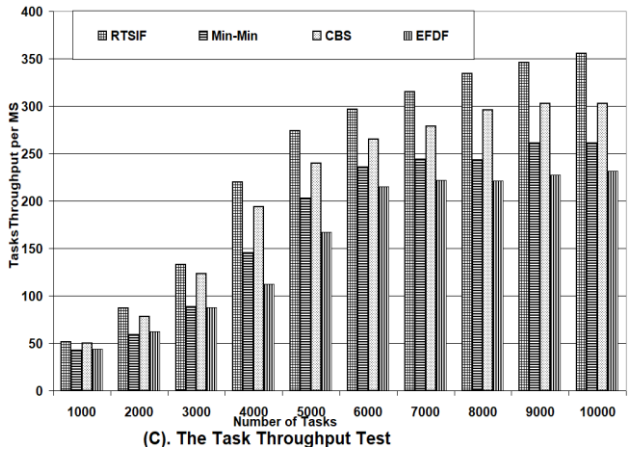
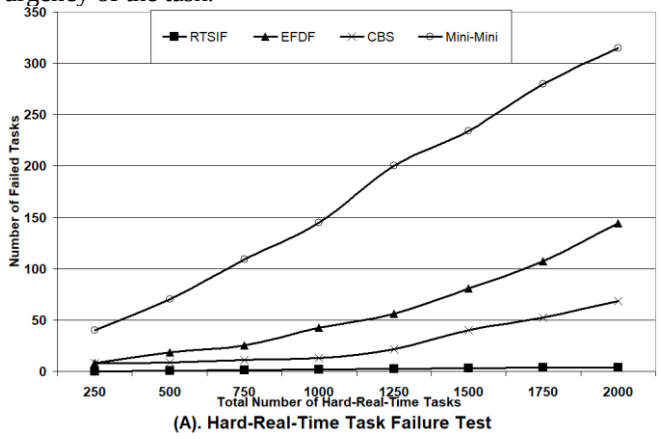
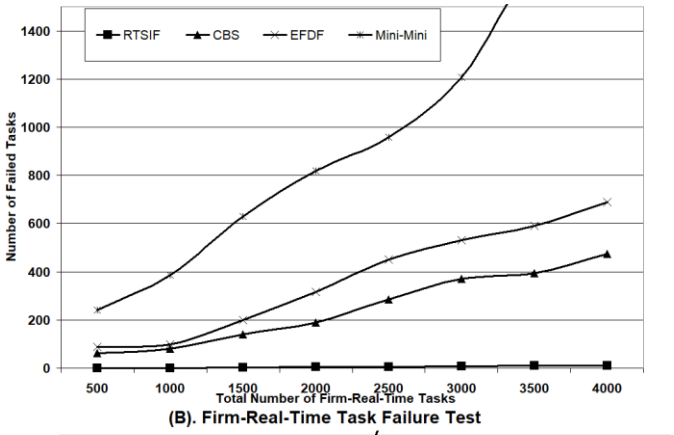


Fig. 4. Performance Comparison using all type of tasks

IV.C) System Persistency measurement

This section measures the persistency of the services in the compared models by measuring the number of failed tasks in each type of real services. Each of value of this test is the average of ten trials. Figures 5 (A, B and C) show the number of failed tasks by increasing the workload. In these figures we notice that; the number of failed tasks in RTSIF has insignificant values. The impressive performance for the RTSIF is attributed to the load balancing between the cluster node in each tier in the system. Moreover, the resource allocation process in RTSIF is based on two factors; the first factor is the urgency of the tasks which represent the raise priority of tasks. The second factor in RTSIF resource allocation strategy is property of the resources. Since the CBS is prioritizing the tasks in the resource allocation process, hence the CBS curve has tasks failure less than the Mini-Mini and EFDF curves. Unfortunately, CBS algorithm doesn't give consideration for the system failure or the level of the resources (mist, fog or cloud). Also, the resource allocation process in Mini -Mini algorithm is based only the size criteria, which is not suitable for real time systems. In another hand, the EFDF algorithm is based only the deadline, which neglecting the urgency of the task.





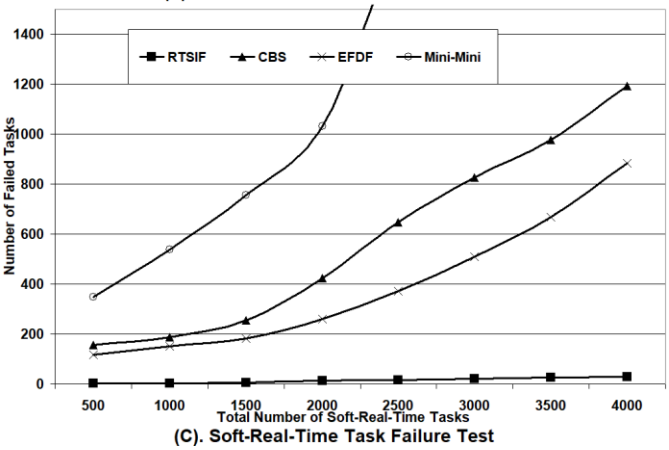


Fig. 5. Persistency measurement for real time tasks

V. CONCLUSION AND FUTURE WORK

In this paper, a model (MLITS) based on Cloud-Fog-Mist architecture is proposed. This model handles and schedules IoT tasks based on their urgency. Several algorithms are introduced inside this model to manage load balance through cloud-fog-mist architecture. Effectively allocating resources with the fault tolerance mechanism for cloud-fog-mist layers was the main goal of the proposed model. The focus was on making the mist layer always available to serve hard tasks that need immediate processing and cannot be delayed, such as the tasks of self-driving cars. Several experiments were conducted to test the proposed model in this research, where all kinds of IoT tasks were addressed. Comparisons between the proposed model and several others are made to compare turnaround, waiting time as well as throughput and persistency. The results showed that the proposed model outperformed all the models that were compared

REFERENCES

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

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

[3] Jalowiczor, J.; Rozhon, J. Voznak, M. (2021). "Study of the Efficiency of Fog Computing in an Optimized LoRaWAN Cloud Architecture. Sensors", 21(9), 3159. https://doi.org/10.3390/s21093159

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

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

[6] Verma, P., & Sood, S. (2018). Fog Assisted-IoT Enabled Patient Health Monitoring in Smart Homes. IEEE Internet of Things Journal, 5, 1789-1796.

[7] Georgios L, Stavrinides, and Helen D. Karatza. (2018) "A hybrid approach to scheduling real-time IoT workflows in fog and cloud environments", Springer Science+Business Media, LLC, part of Springer Nature, 1-17.

[8] Chen, C.Y. (2018). Hasan, M.; Mohan, S. Securing real-time internet-of-things. Sensors, 18, 4356.

[9] Mohammadi, A.; Akl, S.G. (2005). Scheduling Algorithms for Real-Time Systems; Technical Report; School of Computing Queens University: Kingston, ON, Canada.

[10] Satyanarayanan, M.; Simoens, P.; Xiao, Y.; Pillai, P.; Chen, Z.; Ha, K.; Hu, W.; Amos, B. (2015). Edge Analytics in the Internet of Things. IEEE Pervasive Compute. 14, 24–31.

[11] Yassine, A.; Singh, S.; Hossain, M.S.; Muhammad, G. (2019). IoT big data analytics for smart homes with fog and cloud computing. Future Gener. Comput. Syst. 91, 563–573.

[12] Tang, C.; Wei, X.; Zhu, C.; Chen, W.; Rodrigues, J.J. (2018).Towards smart parking based on fog computing. IEEE Access . 6, 70172–70185.

[13] Munir, A.; Kansakar, P.; Khan, S.U.(2017). IFCIoT - Integrated Fog Cloud IoT—A novel architectural paradigm for the future Internet of Things. IEEE Consum. Electron. Mag. 6, 74–82.

[14] Ning, Z.; Huang, J.; Wang, X. (2019). Vehicular Fog Computing: Enabling Real-Time Traffic Management for Smart Cities. IEEE Wirel. Commun. 26, 87–93

[15] Yogi, M.K., Chandrasekhar, K., & Kumar, G. (2017). Mist Computing: Principles, Trends and Future Direction. ArXiv, abs/1709.06927, pp:19-21.

[16] Asif-Ur-Rahman, M., Afsana, F., Mahmud, M., Kaiser, M., Ahmed, M.R., Kaiwartya, O., & James-Taylor. (2019). A. Toward a Heterogeneous Mist, Fog, and Cloud-Based Framework for the Internet of Healthcare Things. IEEE Internet of Things Journal. 6, 4049-4062.

[17] Arkian, H., Diyanat, A., & Pourkhalili, A. (2017). MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. J. Netw. Comput. Appl., 82, 152-165.

[18] Mann, Zoltan. (2021). Notions of architecture in fog computing. Computing. 103. 1-23. 10.1007/s00607-020-00848-z.

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

[20] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, (2011). ―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.

[21] A. Thomasa, Krishnalal Ga, Jagathy Raj V Pb, (2014). "Credit Based Scheduling Agorithm in Cloud Computing Environment", ICICT pp. 913 – 920.

[22] Jagbeer Singh, Bichitrananda Patra, Satyendra Prasad Singh. (2011). "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, 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.