Journal of Communication Sciences and Information Technology (JCSIT)
An International Journal

Resource Allocation Strategy in Fog Computing: Task Scheduling in Fog Computing Systems

Asmaa shoker[1], Mohammed Amoon[1, 2], Ayman M. Bahaa-Eldin[3] and Nirmeen A. El-Bahnasawy.[1]

asmaa.shoker82@gmail.com, mamoon@ksu.edu.sa, ayman.bahaa@eng.asu.edu.eg, and nirmeenA.El-bahnasawy@el-eng.menofia.edu.eg.

[1] Computer Science & Eng. Dept., Faculty of Electronic Eng., Menouf, Egypt.
[2] Dept. of Computer Science, CC, King Saud University, Riyadh 11437, Saudi Arabia.
[3]Misr International University, on leave from Ain Shams University, Cairo, Egypt

Corresponding author: Asmaa shoker ahmed.r.master@gmail.com, asmaa.shoker82@gmail.com

ABSTRACT The fog computing model has attracted considerable research attention, as it concentrates on making cloud-based services more effective and timely for the Internet of Things (IoT) users. The fog layer between the user and the cloud layers is aimed at minimizing transmission, processing time, and total costs. Nevertheless, the use of emerging virtualization technologies in fog planning and resource management was hampered by restricted resources and low-delayed services. This paper offers a new task scheduling algorithm called task priority dynamic implementation (TPDI) based on the priority level in the fog layer to help the rising number of IoT, intelligent devices and to optimize the performance of timely execution and minimize costs. Performance assessment indicates that the proposed algorithm decreases overall response time relative to current task scheduling algorithms. It is critical for emerging brownfield computing technology, which we feel is useful for the priority algorithm for a variety of applications.

Keywords: Fog Computing. Execution Time. Priority Levels. Task Scheduling. Resource Allocation.

I. INTRODUCTION

In many domains, computing is extensively used. Even so, several issues with cloud computing have arisen with the growth of the so-called Internet of Things (IoT). The delay-sensitive and locational computing applications can only be completely compliant with cloud computing in view of the many IoT devices, while the high construction cost, on the other hand, makes cloud computing applications impossible. In the next few years, vast quantities of data will be moved to data centers for processing with the growing number of IoT devices (Cisco expects that by 2020 it will hit 50 billion connected devices and 1 trillion by 2025). When IoT also handles the vast number of devices and data using the existing cloud computing model, this can lead to high delay and network congestion [1].

Fog computing is a virtual network between IoT devices and cloud servers, providing computing and stocking services on the internet's edge. Fog computing consists of computers that have low performance and wide distribution, and are similar to terminal users as a distributed architecture. If the terminal computer of the network raises a request for service, fog computing initially performs data filtering, preprocessing, and analysis. The stored data is then moved to the operating system so that the cloud data center will be less burdensome. The number of mobile devices located at the border of the network is growing rapidly with the growth of IoT technologies. Huge data must also be stored and analyzed to fulfill different user requests. Cloud computing is well suited for data storage and processing, while the central servers are far from end-users, which could lead to large delays. The quality of service will be greatly reduced, particularly in delay-sensitive applications [2].

Huge IoT applications demand the connection of a large range of intelligent equipment to be used in shipping environments, in intelligent homes, in intelligent societies, and in intelligent power systems, etcetera, requiring regular cloud updates that cost little in the end. In this area, the applications need low-cost, low-energy, extended coverage and highly scalable user equipment to ensure the successful deployment of large IoT applications, and critical IoT applications like remote healthcare, traffic control and industrial control (driven robotic vehicles). Generally speaking, the IoT's diverse applications provide endless possibilities and its fullest capacity is only achieved by guaranteeing that smarter devices are linked via the Internet [3].

Fog applications' resource-constrained and latency-sensitive design makes the control of resources one of the most critical tasks in Fog. The decision on the allocation and scheduling of resources is therefore extremely significant. Efficient job planning can provide fast, timely, and desirable responses in intelligent systems. An illustration, a patient's condition needs quick notification in an intelligent healthcare system in order that could save the life of a patient. Consequently, an effective work scheduling algorithm needs to be built to optimize the use of these heterogeneous Fog devices. The ultimate aim is to reduce response time and network usage without rising energy consumption [4].

A big difficulty in communication-wise scheduling is that processed tasks are not supposed to be sent from one queue to the next, as do conventional queuing networks. Timing decisions also decide how activities are routed across the network. It is therefore not clear whether the highest efficiency in these networks can be achieved and what policies are appropriate for their performance [5]. The conventional algorithms can therefore not reduce the time of response for latency-sensitive applications. A scheduling algorithm needs to be built that can reduce average response time and network use in latency-sensitive applications and optimize energy use [4].

This paper presents an algorithm for fog computation based on priority tasks, which ensures that services for various types of applications are delivered in time. The goal is to enhance the use of Fog devices on the network edge and decrease the latency and delay of the application. The style is particularly suitable for very low latency tolerances applications. For efficient allocation of resources, fog nodes in the fog layer may cooperate amongst each other. First, the tasks in the fog layer are assessed depending on the level of priority. Once the fog layers full all of the micro datacenters, tasks are just distributed through the cloud layer.

The content of this paper is organized as follows. Section II summarizes related work, section III presented task scheduling problem, section IV proposed Scheduling Algorithm, section V presented simulation results, section VI presented conclusion.

II. RELATED WORK

We look at a few recent resource management techniques that have been advocated for enhancing fog computing performance. While some of these resource management techniques focus on scheduling jobs on these resource-constrained Fog devices, the majority deal with work allocation. The following objectives are shared by job allocation and scheduling: to improve service quality, to minimize make-span, network utilization, cost, loop delay, device energy consumption, and to maximize network availability, etc. The algorithms have so far only optimized one or two of the parameters of the aforementioned criterion. Here is a brief review of some of these recommended approaches to scheduling and resource allocation distribution.

A dependent job allocation technique to Fog nodes was put out by [6]. Jobs that are dependent on one another require data connection with one another. Workflows or the Directed Acyclic Graph (DAG) are used to depict dependent jobs. By navigating a DAG, the authors determine which tasks are most important and then assign those tasks to Fog nodes. A work is forwarded to the cloud if the fog nodes are unable to complete it because of resource limitations. The authors fail to take into account crucial factors like the budget of the fog supplier and process execution deadline restrictions.

A distributed architecture offered by fog computing at the network's edge enables low-latency access and prompter handling of application requests. With this increased computational power, it will be possible to create new systems for managing and allocating resources that will make the most of the fog infrastructure. This study examines the issue of resource allocation while taking into account the hierarchical architecture made up of cloud data centers and edge capacity, examining application classes, and utilizing various scheduling policies. Where studied a number of scheduling algorithms (Concurrent, FCFS and the delay priority), that take user mobility and edge computing capability into account to overcome this difficulty. This study highlighted that the benefits of combining the application classes with scheduling regulations in scenarios that highlight different scheduling techniques, notably in the setting of user mobility.
It faced some difficulties prioritizing the delay of time-sensitive applications, so the scheduler should be informed of the application requirements by the application classification in order to prioritize cloudlet usage while achieving other objectives like optimizing network usage and cutting down on processing time [7].

Online gaming, smart health, video surveillance, and other real-time applications cannot tolerate the increased latency and bandwidth consumption. To combat the rise in latency and bandwidth consumption in cloud computing, fog computing has evolved. At the edge of a network, fog computing offers storage, processing, networking, and analytics services. Resource-allocation and job-scheduling challenges remain in the fog computing layer due to limited resources. Therefore, choosing the assignment and scheduling of a job on a Fog node is crucial. An effective job scheduling system can speed up application requests' responses while using less energy. To reduce the delays for latency-critical applications, proposed and implemented an efficient job scheduling algorithm (SJF) in [4]. The suggested approach decreases the typical loop time and network utilization by scheduling jobs on Fog devices based on length. Despite lowering the average waiting time, the suggested SJF algorithm can starve tuples with longer lengths.

Our contributions in this paper, In order to improve the scheduling of huge tasks, this section introduced a new technique called Task Priority Dynamic Implementation (TPDI) that is based on application classifications and implementation priorities. The proposed algorithm aims to minimize the completion time needed to execute the task by virtue of the First-Fit allocation approach while maximizing use of the Fog devices that are accessible at a network's edge. , as the processor assigns the nearest memory partition, on the fog node devices as it seeks the nearest suitable server, the execution is very fast. This is done to ensure that the time and costs necessary for the complete fulfillment of the required task are met. This will be explained in detail in section v.

III. Task scheduling problem

Fog computing architecture:

Fog computing is a versatile and cloud-based platform [8] since it links sensors at the end of the network and data center in the cloud. It is a distributed paradigm that supports the creation of cloud services. One such paradigm has the benefit of the features of near the edge from users, cloud resources, and its infrastructure. This approach would allow Internet data and applications to fulfill their wide storage, processing, management, communications and scalability needs locally from advanced user-friendly systems [9], [10].

The key aspect of foggy is the wireless transfer of data to the computers, objects and their application services, which are distributed in the Internet cloud, on one cloud and on one side and on its routing devices, depending on the resource available [11].
The fog computing interface challenges the parties' response to each other between mobile devices and associated cloud, through flexible mobile cloudlets located at the edge of the network [12]. They are mobile mini-data centers with a set of connected computers that provide good Internet connectivity, are rich in resources and are available for use by nearby equipment. By means of which sufficient resources are provided at the edge to support interactive applications that require less transmission time.
Therefore, fog computing is a three-layer structure [13], [14] where the fog intermediates with what it calls all the peripherals, that is called the fog nodes which can be used anywhere through a network connection [15].
As illustrated in [16, Fig 1] demonstrates a fog computing architecture in which three layers consist: the lower layer, which is regarded as a layer on the edge, consists of Internet nodes, including sensors and devices, etc. The middle layer is the switches, gateways, and access point or cloudlet layer [17]. The cloud layer for the top layer comprises servers and data centers.

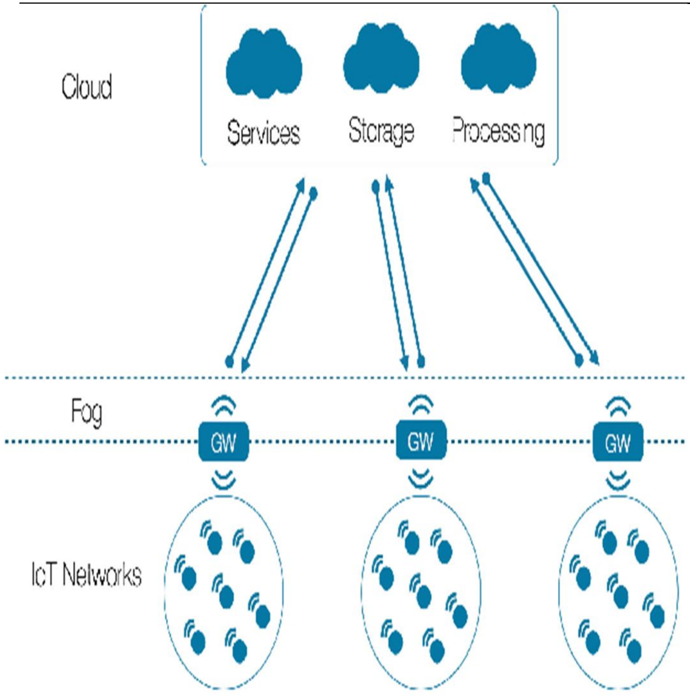

FIGURE 1: Fog computing model.

System model:

In a heterogeneous environment, applications fight for diverse devices' limited resources. At different Fog nodes, these workloads are assigned and carried out. When using the traditional techniques for simple FCFS algorithm task scheduling in fog computing, all jobs have equal priority, which lengthens response times for jobs with brief burst times. Additionally, by allocating jobs to Fog devices based on length, the recommended method reduces typical loop time and network usage. The suggested SJF technique can starve longer tuples despite reducing the average waiting time. The goal of the fog computing paradigm is to speed up responses and decrease network traffic. Consequently, a job scheduling algorithm for Fog must be created and implemented with the following goals:

1. Reduce application loop lag;
2. Limit the use of the network;
3. Utilize the capabilities of Fog gadgets effectively.

We used applied modules in the manner described below to create a case for the suggested approach:

- All application units and IoT devices at the edge are connected to fog nodes on a low-level fog device. It gathers crucial information from application sensors and processes it to generate and show emergency notifications. Additionally, it accepts delay_ tolerant applications data and transfers it to the organizer unit.

- On a high level fog device that receives data from fog nodes, the regulator unit is mounted. Creates a tailored schedule for tasks in response to appointment requests. The scheduler is handling this appointment setting. It sends crucial data for applications to the records database of the organizer.

- Database of tasks (DT). This device is positioned on a cloud. It receives information from the regulator for long-term analysis and storage. Transmits to the organizer patterns for all necessary task states and their priority of execution. [18, Fig 2] illustrates the hierarchical, bi-directional, distributed architecture of fog computing module, which is made up of several levels.

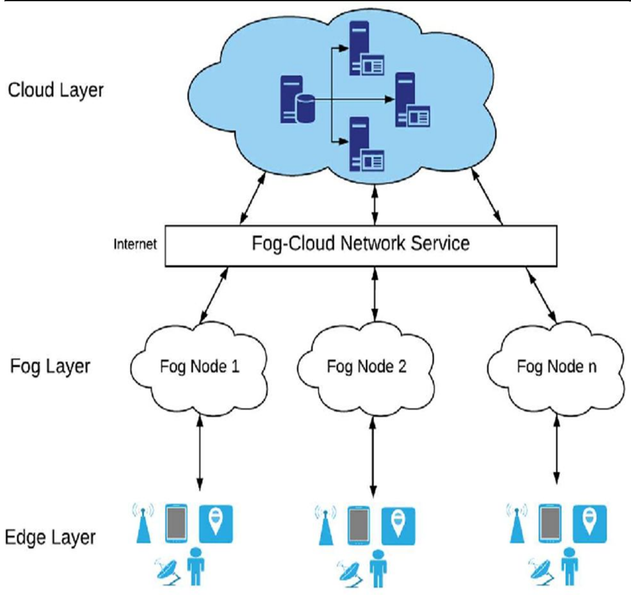

FIGURE 2: Fog computing architecture.

IV. Proposed Scheduling Algorithm:

In order to improve the scheduling of huge tasks, this section introduced a new technique called Task Priority Dynamic Implementation (TPDI) that is based on application classifications and implementation priorities. The proposed algorithm aims to minimize the completion time needed to execute the task by virtue of the First-Fit allocation approach, as the processor assigns the nearest memory partition, on the fog node devices as it seeks the nearest suitable server, the execution is completed quickly. This is done to ensure that the time and costs necessary for the complete fulfillment of the required task are met. The algorithm is presented in the following in the form of pseudo code, followed by a thorough explanation of the method's steps.

The Pseudo code shows the main processes of the proposed algorithm

```
1. Start
2. User submit all applications
3. Set max= Max_priority, h= High_Priority, l=Low_Priority
4. Set priority for each application according to its important
5. Set delay for each application according to the arrival time
6. Sort the applications by descending order according to the priority
7. For each application
8. If priority >h
9. Allocate the application modules to the Fog devices
10. Else if l<priority<h
11. Allocate the large application modules to the fog devices and the rest modules to the cloud
12. Else
13. Allocate all application devices to the cloud data center
14. End for
15. Check if there are new submitted applications go to step 4
16. Else end
```

Symbol *l* refers to requests with lower priority, *h* to requests of higher priority, and *Max* to max requests with priority; this occurs in situations where high priority requests have arrived concurrently, a request is picked depending on the time of arrival.

As seen in the suggested technique, the scheduler checks the implementation priority of the waiting list of tasks, when a fog node gets a group of incoming tasks from a sensor or another fog node. The scheduler allots processing resources to the incoming tuple and continues running if waiting list is empty.

When an incoming task (Ti) is completed, it changes into a finished task (Tf), which is then added to the list of completed tasks as a whole. If the waiting list is not already full, Ti is added to it. The waiting list is then arranged according to its significance. Then, based on the arrival time, the delay is modified for each application. . As a result, the applications were arranged according to priority in declining order. When a request's priority reaches the maximum priority level, fog node devices are assigned.

Once the incoming requests have been assigned in order of priority to the nearest fog node, it is next determined whether all of the resources in the fog

layer are sufficient to meet the amount of requests. Should this not be the case, allocate the major application modules to the fog devices and the remaining modules to the cloud.
When the set is finished, the scheduler chooses the highest priority (Max) set from the top of the waiting list, giving the virtual machine time to finish. Tasks with the highest priority (Tmax) are finished and added to the list of finished full tuples after completion.

V. Simulation Results

Our scheduling strategy is predicated on the idea that there are two categories of applications: critical and delay-tolerant. The simulator continuously generates traffic needs from user devices. These traffic demands are realized and distributed to Vms in the fog nodes according to priority levels. This was achieved using performance indicators like the overall time spent allocating tasks to servers, the typical waiting time, and the typical end time.
Assume that (t_1, t_2 … Tn) denote the number of tasks arranged in descending order, according to priority in a temporary queue (referred to as TQ).
pri, i=1, 2, 3… n, as a priority of a process, the submitted processes are ordered in the ready queue (referred to as RQ). The processes arranged in a descending manner according to their priority.
The tasks are distributed on VMs based on two conditions:
• The first condition: It is the nearest or first server.
• The second condition: that he must have sufficient space or resources according to the type of task.
Set of tasks T= {t_1, t_2… t_n}.
Set of VMs P= {p_1, p_2... p_m}.

$$priority\ for\ each\ task = \begin{cases} Exec(Ti), & 1 \le i \le n, \quad non_{equal} execution\ time. \\ Arival\ (Ti), & 1 \le i \le n, \quad equal\ execution\ time. \end{cases}$$

For execution our proposed algorithm, which should be running when a new application request arrives in a fog layer, and decide where it should be running: on the fog or cloud. Therefore, the following details the steps to implement the proposed algorithm and the related working SJF algorithm.

<u>For our algorithm:</u>
1- Sort tasks by priority.
2- Loop through sorted tasks.
3- For every task:
4- Check if there is Network Node has the resource equal to or big than the task resource required.
5. If true: attach task on the server by first-fit.
5.1 get first Network Node has the resource equal or big than task resource required.
5.2 attach the task to the virtual machine.
5.2.1 Set task as attached to network node.
5.2.1 Reduce the available resource of network node.
6- If false.
6.1 Set task as waiting.
6.2 Go to step 3.

<u>For SJF algorithm</u>:
1- Get tasks by types.
2- Sort tasks by length.
3- Loop through sorted tasks.
4- for every task:
5- Check if there is Network Node has the resource equal or big than task resource required.
6. If true: attach task on server by best-fit.
6.1 loop through network nodes.
6.2 get best network node has available resource has the resource equal or big than task resource required.
6.3 Attach the task to the network node.
6.3.1 Set task as attached to network node.
6.3.2 Reduce the available resource of network node.
7- If false.
7.1 Set task as Waiting.
7.2 Go to step 4.

Simulation results are presented in this section for evaluating the efficiency of the proposed algorithm. A simulator had been built using visual C# .NET 4.7 on a machine with Intel(R) Core(TM) i3 CPU M 350 @2.27GHz, RAM of 8.00 GB, and the operating system is window 10, 64-bit. With the framework asp.net mvc, database sql server, using visual studio 2020 and Microsoft sql server management studio 18.8.

The results from the TPDI algorithm are compared to the SJF algorithm [4] based on the aforementioned metrics, by the software spawned that generates a large number of random tasks, by implementing up to 600 tasks. The performance distinctions of the total time of distribute tasks on servers, the average waiting time, and the average end time are shown in figures 3, 4, and 5, respectively, in the same order.

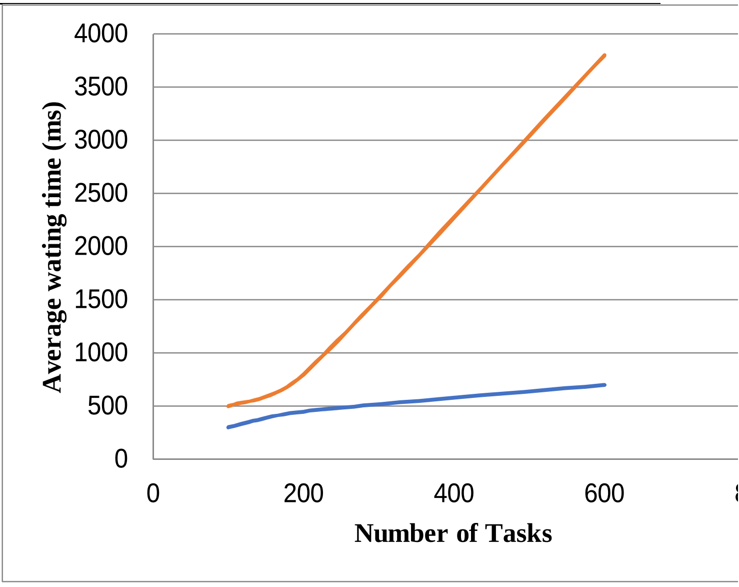

FIGURE 4: the average waiting time.

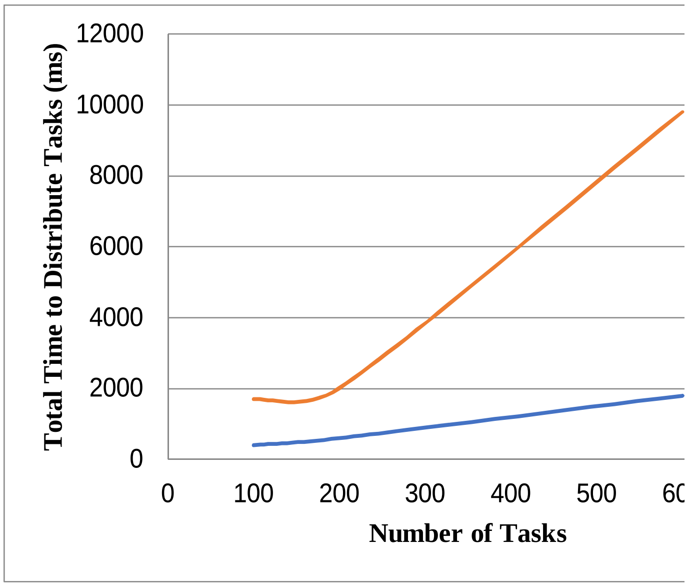

FIGURE 3: the total time of distribute tasks on servers.

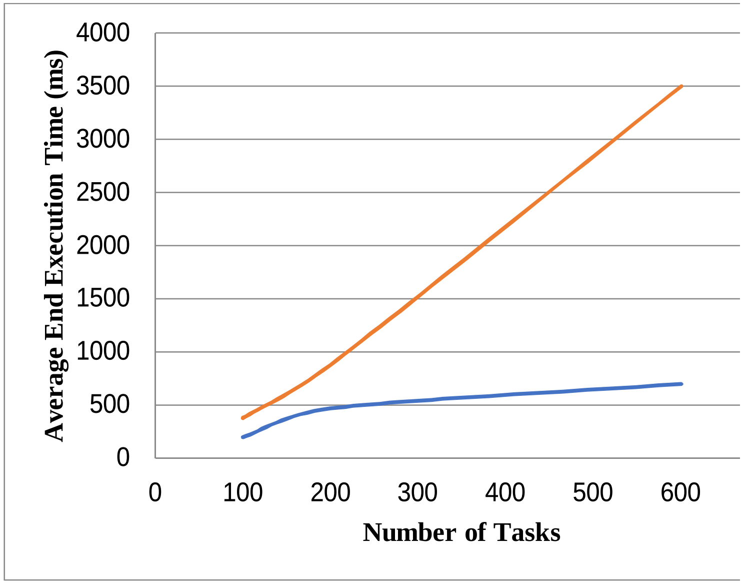

FIGURE 5: the average end execution time.

Additionally, a new simulation has been added to demonstrate the efficiency of the proposed experimental TPDI algorithm, compared to the traditional FCFS algorithm. This was done by evaluating the output of the proposed algorithm based on the results of the ifogsim simulation [17]. The experimental analysis of the fog environment assists in evaluating and developing ideas, also in measuring the effectiveness of policy and technology on resources management in transition times, network congestion, energy consumption, and cost. The simulation of the computer environment is very useful since the actual implementation of the fog environment for analysis is very costly because there are many nodes of fog and data and Internet devices the enormous things all, in addition to the cloud data, also it offers mechanisms for the design and evaluation of a personalized experience environment, which also helps in the assessment of redundancy. Therefore, a comprehensive study was presented to evaluate through the use of performance metrics (energy consumption, total network usage, execution time, and cost of implementation time in the cloud).

Below are results of the comparative, which show that prioritizing minor delays applications would increase application efficiency compared to standard resource sharing techniques such as FCFS, to describe how resource planning in fog computing can improve applications by taking into account geo-location and various classes of applications. We identify two classes of applications: delay-sensitive such as electroencephalography (EEG) tractor beam game, and a delay-tolerant such as a video surveillance/object tracking application [7]. Each application of them contains a number of modules, which the same application modules shall be grouped to be placed on the same device. The EEGTBG application has 5 modules (EEG sensor, client display, concentration calculator, and coordinator), the video surveillance/object tracking application having 6 modules (camera, object detector, motion detector, object tracker, (PTZ) control and user interface. Therefore, a module processing loop must be completed for each application (the time it takes to implement the application loop) to display the results.
Therefore, once the emulator produces frequent batches of traffic requests from the user's devices, both algorithms begin to be executed, and the results for the runtime are shown in Fig 6 by gradually increasing the number of requests.

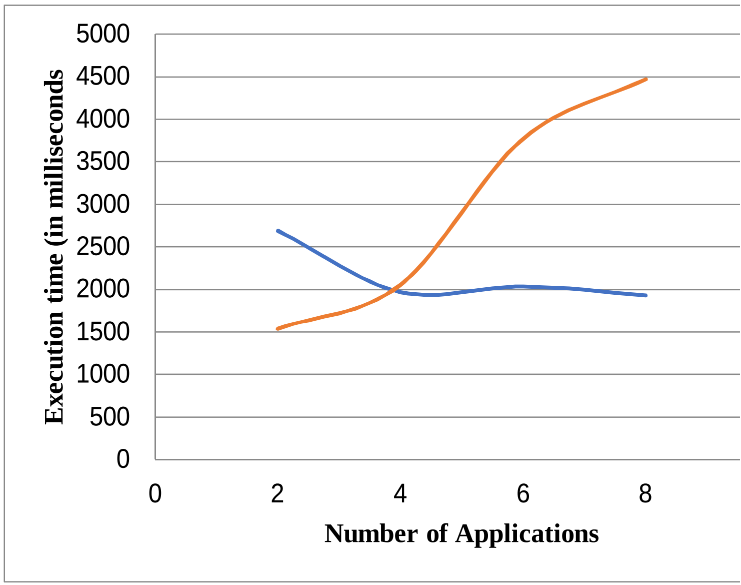

FIGURE 6: Execution time of the applications.

The increased use of network services raises the number of devices, which contribute to network congestion. Ultimately, this congestion leads to poor results on a Cloud-based app. Fog computing helps to reduce network congestion by spreading the load on Fog nodes. The comparison between total network use of TPDI and FCFS algorithm is shown in Figure 7. The number of applications is displayed over x-axis, while an average network use is represented along y-axis for 120000 simulation times. The result showed that the total network usage using TPDI algorithm can significantly reduce when the number of applications is below 4 and, but it gradually increases with the increase in the number of applications at the peak time compare to the FCFS algorithm.

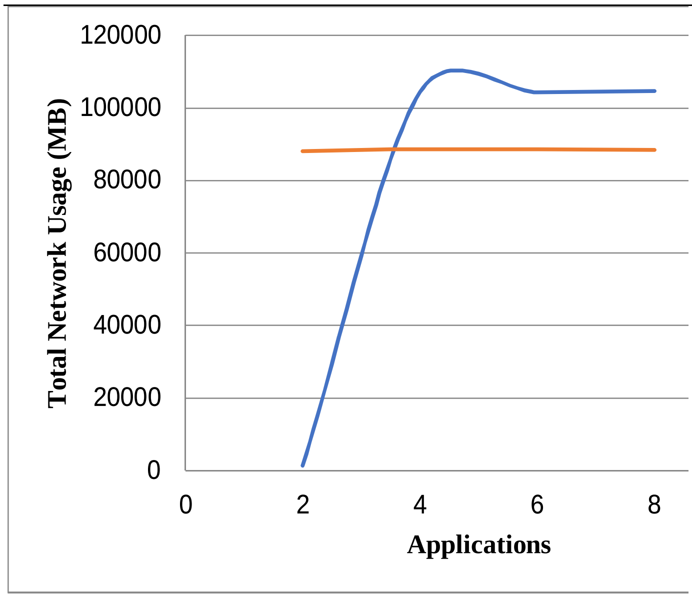

FIGURE 7: Total Network Usage.

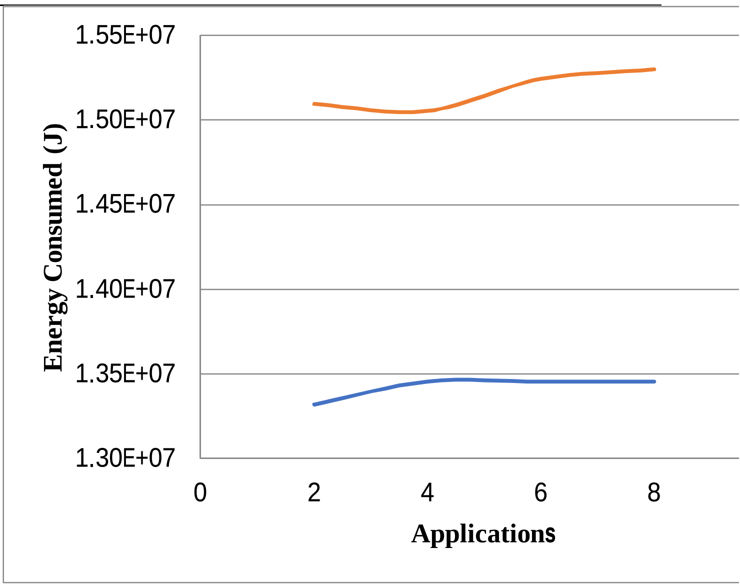

FIGURE 8: energy consumed TPDI vs. FCFS.

The computations in both the fog layer and the cloud layer, data forwarding that requires real-time, and the low latency services, which are handled at the level of the fog computing, are among the factors responsible for energy consumption, figure 8 shows the average energy consumption of TPDI versus FCFS, finally figure 9 shows the cost of task execution.

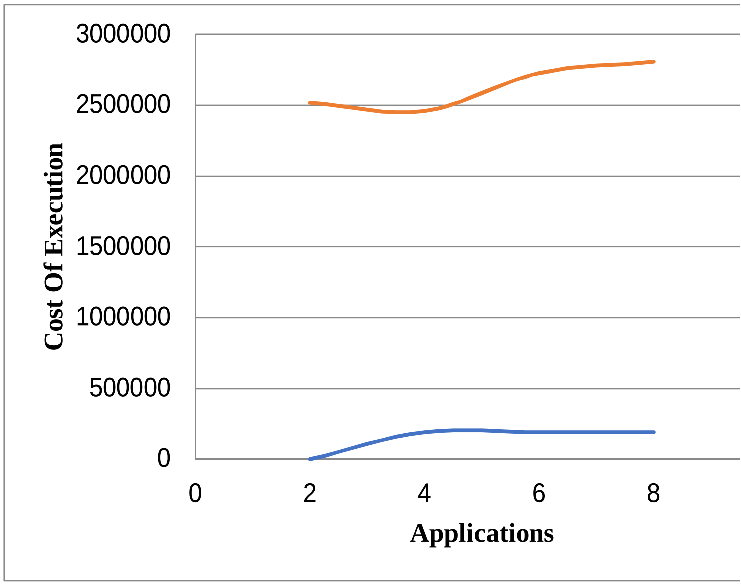

FIGURE 9: cost of task execution.

VI. Conclusion

Fog computing is a valuable extension of cloud computing as cloud computing is unable to handle large data in real-time due to some limitations and challenges by smart devices, routers, and

actuators. Cisco, therefore, implemented computing fog. It concentrated on the issues and problems currently facing cloud computing. In this paper, we have highlighted cases of the use of fog in various applications with their classifications and differentiate them in processing speed from computer computing. Therefore, we suggested a new TPDI algorithm; its work is based on level priority and classifications of applications. The proposed algorithm has emphasized excellent performance in fog computing layers in terms of time reduction rates for task performance and compared to the advanced SJF algorithm.
In addition, the findings of our algorithm have shown that the proposed algorithm is helpful in the successful management of the resources, as opposed to the traditional algorithms, where the rates showed that the average power consumption is much lower than conventional FCFS, and also improved performance in reducing execution time, network usage, and cost much lower when compared with FCFS.

REFERENCES

[1] L. Yin, J. Luo, and H. Luo, "Tasks Scheduling and Resource Allocation in Fog Computing Based on Containers for Smart Manufacturing," *IEEE Trans. Ind. Informatics*, 2018, vol. 14, no. 10, pp. 4712–4721, doi: 10.1109/TII.2018.2851241.
[2] G. Li, Y. Liu, J. Wu, D. Lin, and S. Zhao, "Methods of resource scheduling based on optimized fuzzy clustering in fog computing," *Sensors (Switzerland)*, 2019 vol. 19, no. 9, doi: 10.3390/s19092122.
[3] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz, "A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges," *IEEE Access*, 2017, vol. 6, pp. 3619–3647, doi: 10.1109/ACCESS.2017.2779844.
[4] B. Jamil, M. Shojafar, I. Ahmed, A. Ullah, K. Munir, and H. Ijaz, "A job scheduling algorithm for delay and performance optimization in fog computing," *Concurr. Comput.* , 2020, vol. 32, no. 7, pp. 1–13, doi: 10.1002/cpe.5581.
[5] C. S. Yang, A. S. Avestimehr, and R. Pedarsani, "Communication-Aware Scheduling of Serial Tasks for Dispersed Computing," *IEEE Int. Symp. Inf. Theory - Proc.*, 2018, vol. 2018-June, pp. 1226–1230,, doi: 10.1109/ISIT.2018.8437763.
[6] X.-Q. Pham and E.-N. Huh, "Towards task scheduling in a cloud-fog computing system," pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7737240/authors#authors
[7] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-Aware Application Scheduling in Fog Computing," *IEEE Cloud Comput.*, 2017 vol. 4, no. 2, pp. 26–35, doi: 10.1109/MCC.2017.27.
[8] S. Sarkar and S. Misra, "Theoretical modelling of fog computing: A green computing paradigm to support IoT applications," *IET Networks*, 2016, vol. 5, no. 2, pp. 23–29, doi: 10.1049/iet-net.2015.0034.
[9] A. V. Dastjerdi and R. Buyya, "Fog Computing: Helping the Internet of Things Realize Its Potential," *Computer (Long. Beach. Calif).*, 2016, vol. 49, no. 8, pp. 112–116, , doi: 10.1109/MC.2016.245.
[10] M. Aazam and E. N. Huh, "Fog Computing: The Cloud-IoT/IoE Middleware Paradigm," *IEEE Potentials*, 2016, vol. 35, no. 3, pp. 40–44, , doi: 10.1109/MPOT.2015.2456213.
[11] K. C. Okafor, I. E. Achumba, G. A. Chukwudebe, and G. C. Ononiwu, "Leveraging Fog Computing for Scalable IoT Datacenter Using Spine-Leaf Network Topology," *J. Electr. Comput. Eng.*, 2017, vol., doi: 10.1155/2017/2363240.
[12] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for Internet of Things: a primer," *Digit. Commun. Networks*, 2018, vol. 4, no. 2, pp. 77–86, doi: 10.1016/j.dcan.2017.07.001.
[13] S. Kabirzadeh, D. Rahbari, and M. Nickray,

“A hyper heuristic algorithm for scheduling of fog networks,” *Conf. Open Innov. Assoc. Fruct*, 2018, pp. 148–155, doi: 10.23919/FRUCT.2017.8250177.

[14] P. Hall and H. Miller, “Sequential, bottom-up variable selection for high-dimensional classification,” *Aust. New Zeal. J. Stat.*, 2010, vol. 52, no. 4, pp. 403–421, , doi: 10.1111/j.1467-842X.2010.00594.x.

[15] H. F. Atlam, R. J. Walters, and G. B. Wills, “Fog computing and the internet of things: A review,” *Big Data Cogn. Comput.*, 2018, vol. 2, no. 2, pp. 1–18, , doi: 10.3390/bdcc2020010.

[16] L. Belli, S. Cirani, G. Ferrari, L. Melegari, and M. Picone, “A graph-based cloud architecture for big stream real-time applications in the internet of things,” *Commun. Comput. Inf. Sci.*, vol. 508, no. September 2017, pp. 91–105, doi: 10.1007/978-3-319-14886-1_10.

[17] H. A. M. Name, F. O. Oladipo, and E. Ariwa, “User mobility and resource scheduling and management in fog computing to support IoT devices,” *7th Int. Conf. Innov. Comput. Technol. INTECH 2017*, pp. 191–196, doi: 10.1109/INTECH.2017.8102447.

[18] R. Neware and U. Shrawankar, “Fog Computing Architecture, Applications and Security Issues,” *Int. J. Fog Comput.*, 2019, vol. 3, no. 1, pp. 75–105, doi: 10.4018/ijfc.2020010105.