Analysis of rank-based latency aware fog scheduling using validating internet of things at large scales

J. Geetha, Shaguftha Zuveria Kottur, Riya Ganiga, D. S. Jayalakshmi, Tallapalli Surabhi
Department of Computer Science and Engineering, M. S. Ramaiah Institute of Technology, Bangalore, India

Article Info

Article history:

Received Aug 6, 2021
Revised Mar 8, 2022
Accepted Mar 22, 2022

Keywords:

Fog computing
Internet of things
Latency
Rank based scheduling
VIoLET

ABSTRACT

With the increase in internet of things (IoT) applications' range and scale, it is essential to test the applications before deploying them in the real world. Most common approaches utilize simulations and testbeds; however, these methods lack real-time failure scenarios and the capability to scale, respectively. A virtual environment is a suitable approach that overcomes these drawbacks further, IoT applications using cloud computing have evolved to shift some computing and storage capabilities to the edge networks for ensuring adherence to strict latency constraints for real-time applications. This led to the emergence of fog computing which provides lower latency and better security, among other advantages. As for any processing tasks, scheduling becomes a critical concern for matching the tasks with the devices having appropriate resources. This paper analyzes a hybridized fog scheduling algorithm based on a ranking approach considering latency as the main parameter. It builds a software layer for scheduling on top of the validating internet of things at large scales (VIoLET) infrastructure. The results are compared with the round-robin scheduling algorithm, and it is found that the hybridized algorithm provides closer actual latency values to the expected one.

Corresponding Author:

J. Geetha
Department of Computer Science and Engineering, M. S. Ramaiah Institute of Technology
Bangalore, India
Email: geetha@msrit.edu

1. INTRODUCTION

Internet of things (IoT) is a network of computable devices that are connected via the internet. This has enabled sharing of information collected through such devices and performing computations on the collected data on the cloud. The cloud is utilized as remote storage and computational platform and is one of the significant enablers of IoT on a large scale which has allowed IoT to be used for a wide range of applications such as smart devices, smart grids [1]-[3], smart campuses [4], [5], and smart supply chains [6], [7]. It allows devices to interact, collaborate and share their data over the internet, increasing connectivity and intelligence.

There are many security concerns related to the use of IoT [8], [9] and the communication of data. Integrating fog computing alleviates some of these concerns [10], [11] as it acts as an added layer of security between the server and IoT devices. Since the fog network can monitor smaller and simpler devices, it can be relied upon to handle the bulk of the security responsibilities. It also allows monitoring of distributed systems and can respond to breaches without shutting down essential system elements. Thus, there are many advantages of integrating fog computing into real-time IoT applications [12].

Fog scheduling also performs a key role in reducing latency and energy consumption by allocating resources efficiently [13]-[15]. Depending on the considerations like homogeneity of the fog environment, ability to share communication and computational bandwidth [16], and the parameters for scheduling, the scheduling algorithms vary in their methodologies.

Validating internet of things at large scale (VIoLET) provides a virtual environment for testing IoT deployments which overcomes the drawbacks of simulation environments and testbeds, namely scalability and lack of real-time failure scenarios [1]. Currently, VIoLET provides only the virtual IoT network and does not have the functionality for applications over IoT, which becomes the users' responsibility. VIoLET lacks efficient scheduling strategies for testing applications over IoT. So, the paper focuses on analyzing a hybridized fog scheduling algorithm based on a ranking approach considering latency as the parameter of importance and it uses VIoLET to do so. The objectives of this paper are i) to review fog scheduling algorithms and practices, ii) to implement the proposed algorithm over the virtual hardware architecture of VIoLET, and iii) analyze the performance of the algorithm.

2. RESEARCH METHOD

2.1. Literature survey

Maximal energy-efficient task scheduling (MEETS) can be used to examine the trade-off association between energy cost and performance gain [17]. MEETS is an algorithm for task scheduling that considers energy consumption due to circuits, computation of fog, and offloading energy. Clouds are logical partitions of cloudlets that are partially ordered, with each division having an upper and lower bound that is thought to affect the rank of cloudlets by n divisions. Depending on the job requirements and the bounds of each division, the scheduler selects the appropriate cloudlet for task assignment [18].

In a heterogeneous fog computing environment, dispersive stable task scheduling (DATS) provides a scheduling technique. Tasks can be offloaded via distributed computing while being executed concurrently by the cloud and various dispersed fog nodes in a broad multi-user paradigm. The performance of DATs is compared to that of iterative and random task scheduling [19].

The environment of distributed computing has changed dramatically as a result of fog computing. New components have been added to the Storm architecture to enable a distributed, quality of service (QoS)-aware scheduler and self-adaptation capabilities. The custom scheduler outperforms the usual Storm, improves application performance, and increases adaptability [20].

Network congestion, minimal bandwidth consumption, security, and fault tolerance are all difficulties that cloud computing faces. This paper's architecture is divided into three layers: cloud, fog, and client. The fog layer is made up of Fog Server Masters and Coprocessor Servers that are spread across numerous locations [21].

Rahbari and Nickray [22] point out that storage, processing power, and cloud latency are all issues that the wireless sensor network (WSN) (IoT infrastructure) faces. Fog computing (FC) reduces bandwidth usage and delivers data to clients rapidly, which is critical for applications. Fog devices (FDs) are small processing units that can execute resource management algorithms and are located near edge client sensors. A knapsack-based greedy scheduling strategy is given in this paper for mapping computer resources to fog network modules. IoT applications are organised as modules in the fog network. A module in FD conducts data processing operations including applying data labels and removing unnecessary items. Microdata centres provide resources to application modules.

On distributed clusters, a technique to decrease inter-fog computing-based radio access points (FAP) interference [23]. FAPs' signal processing and complete cache usage capabilities reduce the front-end load. The proposed scheme achieved over 94% throughput and greatly beat the baseline scheme Fog is a hyper-heuristic algorithm that is used in fog networks to schedule and distribute resources. It has a number of advantages, including reduced latency time, reduced network traffic, and increased energy efficiency. The fundamental concept is to combine the benefits and compensate for the shortcomings of single heuristics [24].

In iFogsim, a knapsack-based scheduling algorithm was optimised as a standard test technique for Fog computing by symbiotic organisms. Symbiotic organisms Search is based on two paired relationships of organisms that are easily found across the environment in three steps: mutualism, commensalism, and parasitism [25]. They can accommodate a larger number of IoT devices thanks to energy minimization scheduling (EMS) [26]. Idle energy first (IEF) and dealing energy first (DEF) are two main EMS techniques (WEF). Except for ILP solution, all methods are performed in under 0.1 seconds for various workloads. Suggest a to enable responsibility for QoS, job classification, resource scheduling, and allocation functions have been established. This is accomplished by scheduling computing resources and classifying IoT applications. Based on QoS criteria and resource constraints, a resource allocation plan is created [27].

2.2. Proposed system

During the literature review, Algorithms 1 for Fog scheduling were studied, and their assumptions were analyzed. The scenarios considered homogeneous or heterogeneous fog environments, computational offloading, sharing communication and computation spectrum, and prior energy consumption knowledge. The scenario considered for this paper was a homogeneous environment without computational offloading or shareable spectrum for communication and computation and with prior knowledge about the requirements of the task. The user can define how many tasks are required for scheduling, and an arbitrary latency requirement is assigned to those tasks. This simulates the latency constraints for real-time applications. A Coremark set of programs are used as workloads. The scheduling algorithm matches the task to the division with the appropriate latency thresholds, and then the device which has the lowest coremark value and is not currently executing another task is selected. If no such device is available, the task is queued. After task execution, the latency for the task is measured, and this is used to update the score of the device using harmonic mean.

Algorithm 1. Bootstrap scheduler

```
Input: number of divisions
Output: devices assigned to each division
If number_of_divisions are null or non-positive or greater than number_of_fog_devices, then
        returnerror_message
end if
read device_listfrom deployment_output. json
read vm_detailsfrom vm_config. json
initialize max and min latency to default values
for each device in device_list do
        connect to the host_vmwhich has the device via SSH
        execute standard task on the device
        measure the latency for task execution
        measure coremark value for each device
        update max and min latency measured
end for
divide the range between max and min latency into number_of_divisionsintervals
sort devices in device_list according to coremark value
for each device in device_list do
        check under which division threshold it falls
        assign it to the threshold and update division_details
end for
store device_detailswith latency and coremark value
store division_detailswith thresholds and assigned device_list
```

Initially, a bootstrap scheduler as per the Algorithm 1 listed above is programmed with the number of divisions to which the fog devices in the architecture will be assigned. The algorithm takes the number of divisions as command line input from the user and initializes the divisions with latency thresholds, and assigns fog devices to them. It also stores division details with threshold and assigned device list. The division thresholds are calculated dynamically using the execution of a standard task/workload, and this is also utilized to assign an initial score to each device which is used to allocate it to divisions. The parameter of importance according to which the thresholds are created depends on the requirement, but for the paper, it is considered as the latency, keeping in mind strict latency requirements necessary for real-time applications. A ranking algorithm, as shown in Algorithm 2 is designed to run in the background at specified time intervals. It recalculates the score of each device and reassigns devices to divisions if required. The frequency of execution of this algorithm depends on the user.

Algorithm 2. Ranking algorithm

```
Input: number of tasks
Output: latency and division details
read device_details which has latency and coremark values for fog devices
read division_details which has threshold value and assigned devices
for each device in device_details do
        if the device_latency does not lie between the assigned division_thresholds then
                if the division has the lowest threshold and device_latency is lower or
                higher threshold and device_latency is higher then
                        modify the threshold to include the latency of the device
                else
                        reassign device to another division
                end if
        end if
end for
update division details
```

The ranking algorithm is used in the proposed latency optimizing scheduler. The algorithm is described in Algorithm 3. The proposed scheduler accepts the number of tasks from the user, schedules and executes them on the fog devices, and updates device latency information.

Algorithm 3. Latency optimizing scheduler

```
Input: number_of_tasks
Output: latency values
read device_detailswhich has latency and coremark values for fog devices.
read division_detailswhich has threshold value and assigned device
read vm_details

ifnumber_of_tasks are non positive integer then
        returnerror_message
end if
for each task in tasks do
        assign arbitrary latency requirements
        determine division with appropriate thresholds
        for each device in division sorted in ascending order of coremark do
                ifdevice is free, then
                        assign task
                else ifall devices are occupied, then
                        queue task
                end if
        end for
end for
consolidate the schedule for all tasks
for each task in tasks do
        connect to host_vm of assigned device
        execute task on device
        update latency detail of device using harmonic mean
end for
display rank algorithms latency values
for each task in tasks do
        assign to devices in round robin fashion
        connect to host_vm of assigned device
        execute task on device
        measure latency
end for
display round robin latency values
```

2.3. Experimental setup and implementation

2.3.1. Virtual environment

VIoLET is used to examine the proposed scheduling method. VIoLET is a large-scale virtual environment for defining and launching large-scale internet of things deployments in could virtual machine (VMs) [1]. It provides a declarative model for specifying Docker-based compute resources that match the performance of native edge, fog, and cloud devices. VIoLET is simple to set up and use, balancing ease and flexibility. It sets bandwidth and latency restrictions for containers and makes it simple to define various network topologies. VIoLET is based on cloud VMs and hence has the ability to scale to hundreds or thousands of devices. It enables the creation of virtual sensors that generate data from a variety of distributions within containers. This provides firsthand knowledge of the user's performance, scalability, and metrics.

VIoLET is a platform built for creating and testing IoT deployments in a scalable virtual environment. It builds the functionality of the hardware devices and their networks and allows the user to deploy any application over the virtual IoT network. Scheduling, an essential aspect in reducing latency and optimizing other specific constraints like energy, is required to ensure that resource allocation is done efficiently. VIoLET utilizes a rank scheduling algorithm to match the incoming task's latency requirements.

The VioLET code repository is available in [1]. Setting up VioLET is relatively more straightforward if a paid account for a cloud service provider is used since it has higher resource availability than free tier accounts, which have restrictions on resources. For this implementation, two Google Cloud platform free tier accounts were used. The basic setup concerning the number of virtual machines used depends upon the IoT deployment that needs to be tested and the coremark values of all the devices in the deployment and the virtual machines themselves. These values can be used to find the number of containers VMs required for the deployment [1].

2.3.2. Algorithm implementation

The proposed latency-aware fog scheduling algorithm was implemented on VIoLET architecture [1], and its performance was compared against the standard round-robin scheduling algorithm. The algorithm implementation is as shown in Figure 1. The Coremark program is used as tasks to be executed on the nodes to ensure a standardized workload.

The IoT deployment was added to the file *infra_gen. json* with the device specifications in *device_types. Json* and *vm_config. Json* respectively. All the VMs should have RSA keys associated with them which can be done by running *ssh-keygen* on the terminals. The Metis graph partitioning does the allocation of the virtual IoT devices to the VMs. Once this is done, Docker should be started on the admin and host VMs. This creates a key-store datastore on the admin VM, which stores the information of the host VMs required for communication amongst them. If this is not done properly, overlay network creation will fail.

Next is running *infra_config.py,* which creates the Docker overlay networks, the device containers, attaches the containers to the networks, and performs the necessary network access setting to operate like an IoT network. This completes the creation of the virtual hardware layer for the IoT deployment, and any application using it needs to be coded separately. If the deployment requires sensors, another file in the code repository [1] addresses this need. It requires that the VIoLET repository be present in all the Docker containers and the presence of specific commands. This can be achieved by adding a Docker file with these specifications on each VM.

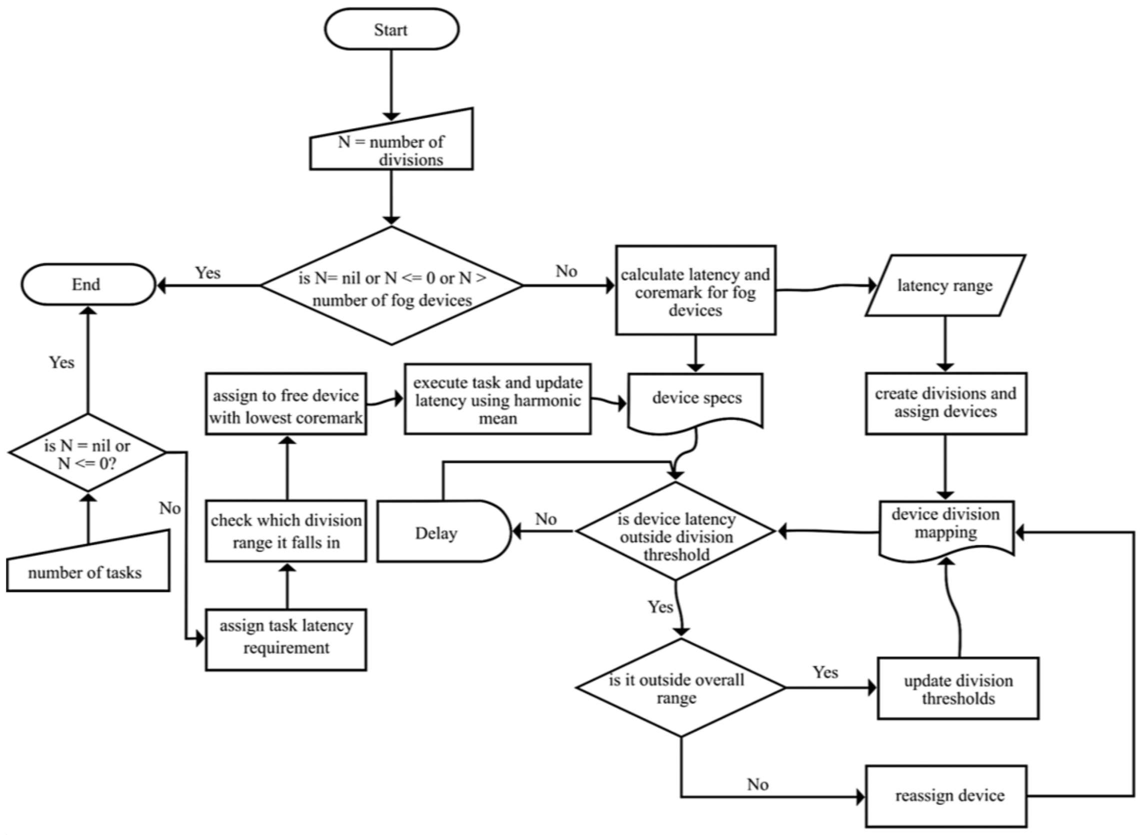
Start
N = number of divisions
is N= nil or N <= 0 or N > number of fog devices
Yes
No
End
calculate latency and coremark for fog devices
latency range
Yes
assign to free device with lowest coremark
execute task and update latency using harmonic mean
device specs
create divisions and assign devices
is N = nil or N <= 0?
No
check which division range it falls in
Delay
No
is device latency outside division threshold
device division mapping
number of tasks
Yes
assign task latency requirement
is it outside overall range
Yes
update division thresholds
No
reassign device

Figure 1. Flowchart of the proposed scheduling algorithm implementation

3. RESULTS AND DISCUSSION

The proposed latency-aware fog scheduling algorithm was implemented on VIoLET architecture, and its performance was compared against the standard round-robin scheduling algorithm. The Coremark program was used as tasks to be executed on the nodes to ensure a standardized workload. The tasks have a required latency threshold according to which the proposed algorithm chooses which division to assign the task to, as shown in Figure 1. The scheduling algorithms are executed serially for the same set of tasks. The measurements are obtained by executing a set of tasks multiple times to reduce the effect of outliers.

3.1. Criteria for evaluation

The main parameter for reduction was the overall latency for the incoming tasks. The focus was on matching the tasks to devices with similar latencies to prevent the non-availability of devices when a task with a really low latency requirement comes in. Another consideration made was to match tasks with the devices, which would result in the lowest energy consumption-because the paper utilized virtual machines for execution, accurate measurements for these criteria are not available and will be discussed in detail in the discussion future work section.

3.2. Results

This section details the findings when two and three divisions were considered for runs of five and ten tasks. A run denotes the execution of multiple tasks. Any measurement made against a run refers to the average value obtained from executing all the tasks in that run.

3.2.1. Number of divisions=2

Figure 2 shows the latency requirements of the tasks versus the achieved latencies by using the proposed and round-robin algorithm. Here N denotes the number of divisions. It can be deduced from Figure 2 that the proposed algorithm does not necessarily achieve lower latency than round-robin algorithm, but the latency achieved by the algorithm is much closer to the required latency of the tasks. This observation is displayed more clearly in Figures 3 and 4, where the standard deviations of the achieved latency of the proposed and round-robin algorithm are shown.

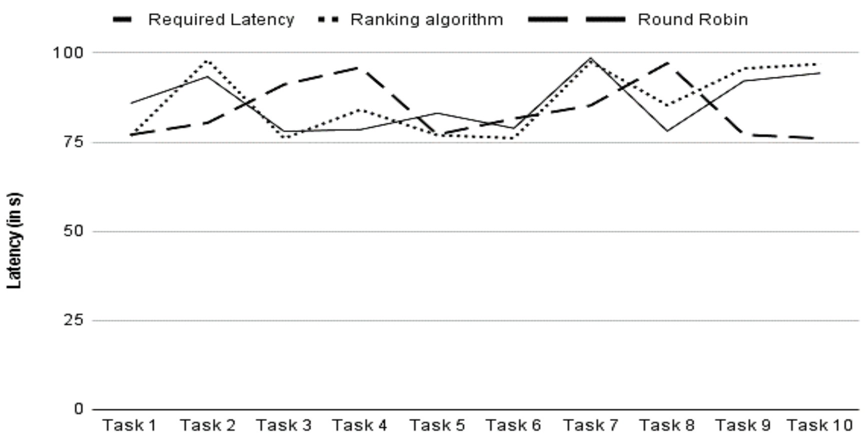

Figure 2. Actual latency vs the required latency for both ranking algorithm and Round Robin for 2 divisions

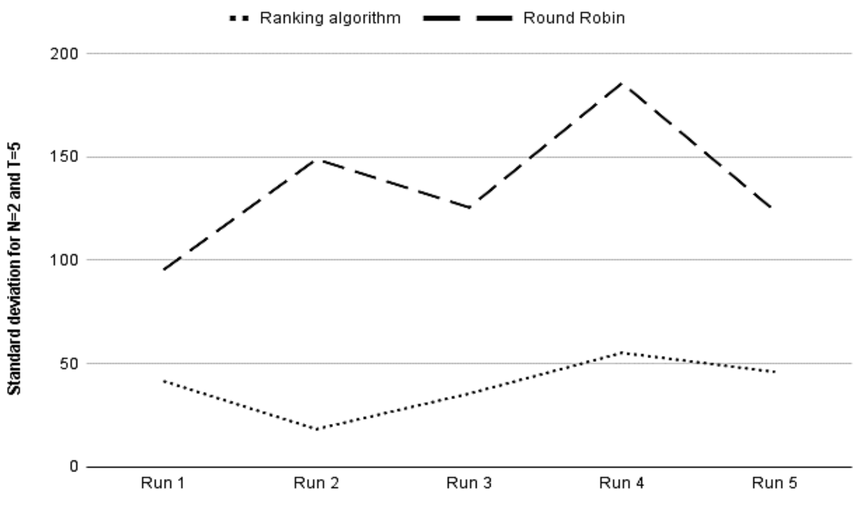

Figure 3. Standard deviation for 2 divisions and 5 tasks is shown for both ranking algorithm and Round Robin

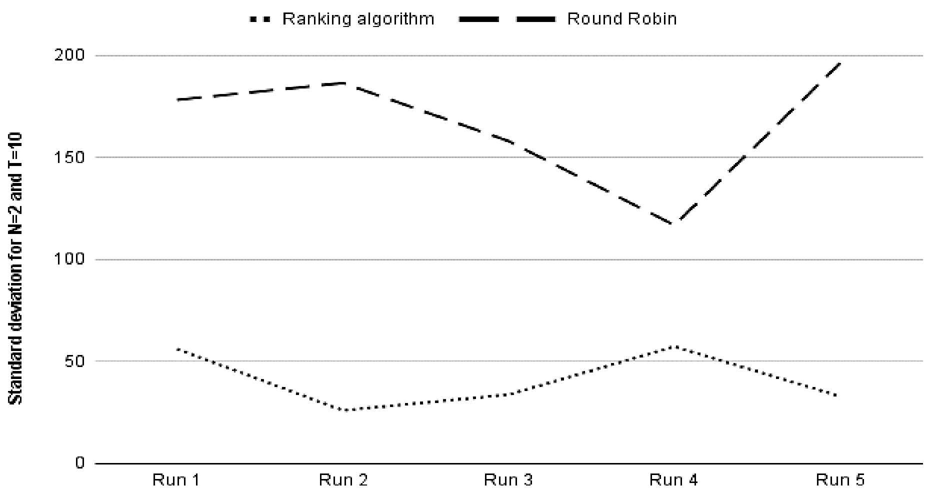

Figure 4. Standard deviation for 2 divisions and 10 tasks is shown for both ranking algorithm and Round Robin

In Figures 3 and 4, the average standard deviation of the proposed algorithm is 71.16% and 75.46% lower than round-robin, respectively. For each run, the average of the task latency requirements and the average of the achieved latencies of a particular algorithm are used to calculate the standard deviation. Thus, it can be observed from Figures 3 and 4 that the proposed latency-aware algorithm matches the tasks' latency requirements more closely compared to the round-robin algorithm.

3.2.2. Number of divisions=3

Figure 5 shows a similar trend to Figure 2, where the required latency of the task is closely matched by the achieved latency of the proposed algorithm compared to round-robin. Figure 6 and Figure 7 show the standard deviation of the achieved latencies with respect to the latency requirements. Figure 6 and Figure 7 that the standard deviation of the proposed algorithm is 74.69% and 66.22% lower than round-robin, respectively, which supports the trend in Figure 2.

It can be seen in Figure 6 that for the last run, the measured values are almost the same, which can also be observed for tasks 8, 9, and 10 in Figure 5. This highlights the fact that round-robin is a randomized algorithm which in some instances, can perform as well as or even better than the proposed algorithm, but as observed, the proposed latency aware rank-based algorithm generally outperforms Round Robin. The focus on decreasing the standard deviation is to reduce the probability that when a task with a low/strict latency requirement enters the environment, the node that can satisfy that requirement is not busy with a task that can handle larger delays.

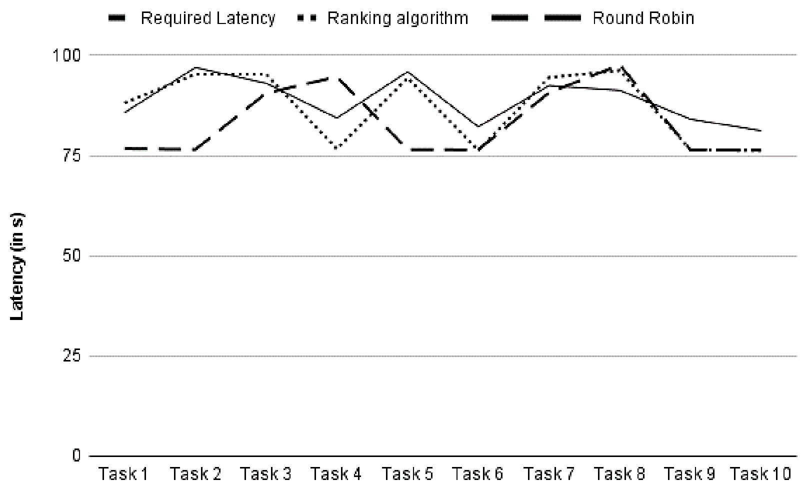

Figure 5. Actual latency vs the required latency for both ranking algorithm and Round Robin for 3 divisions

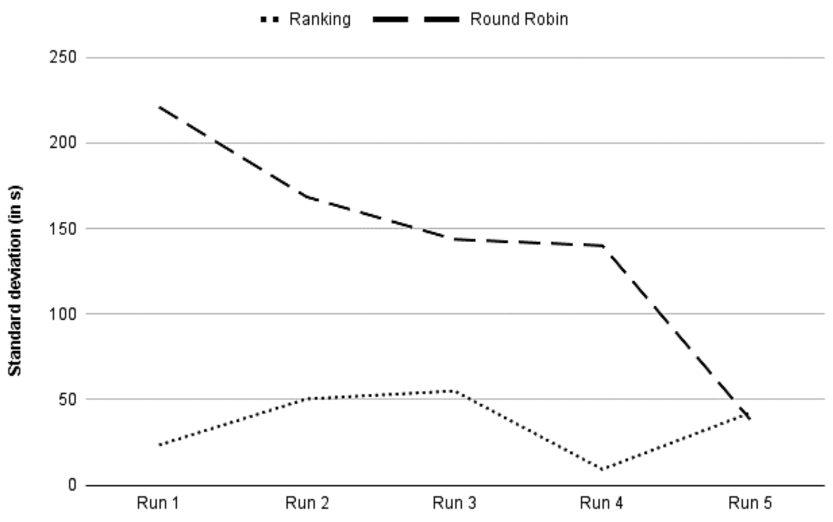

Figure 6. Standard deviation for 3 divisions and 5 tasks is shown for both ranking algorithm and Round Robin

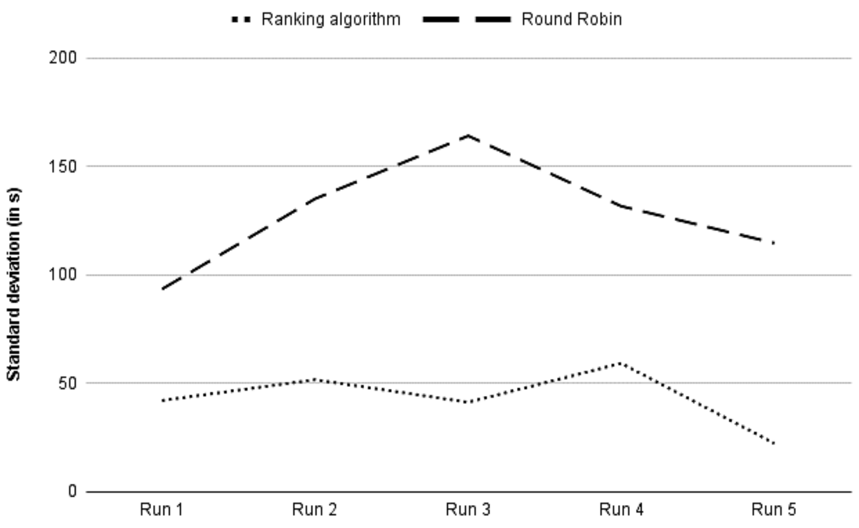

Figure 7. Standard deviation for 3 divisions and 10 tasks is shown for both ranking algorithm and Round Robin

4. CONCLUSION

When it comes to real-time IoT applications, fog scheduling can help reduce latency. Because processing is done closer to the edge device rather than entirely on the cloud, which would increase the delay, it allows co/mputation to be offloaded from resource-constrained edge devices. There are a variety of techniques for both homogeneous and heterogeneous fog settings that allow work offloading to neighbouring nodes based on their bandwidths, especially in heterogeneous environments. Based on the incoming process, traditional optimization techniques such as Knapsack, stable matching theory, and heuristic algorithms such as genetic algorithms and ant colony optimization are used. Another popular solution is to provide architectural layers that assist in assigning incoming workflow to the appropriate previously categorised fog instances based on their resource bandwidth. Based on the characteristics and limits of the installed application. The proposed algorithm is a rank-based latency-aware algorithm that enables a software scheduling layer on top of the VIoLET architecture. The results show that it performs a better job at matching the tasks to devices in terms of latency requirements but may not reduce the overall latency for a set of tasks when compared to an algorithm like round-robin, which does not consider latency as a parameter for scheduling, possibly due to randomness.

REFERENCES

[1] S. Baheti, S. Badiger, and Y. Simmhan, “VIoLET: An emulation environment for validating IoT deployments at large scales,” *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 3, pp. 1–39, Jul. 2021, doi: 10.1145/3446346.

[2] Y. Li, X. Cheng, Y. Cao, D. Wang, and L. Yang, “Smart choice for the smart grid: Narrowband internet of things (NB-IoT),” *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1505–1515, Jun. 2018, doi: 10.1109/JIOT.2017.2781251.

[3] F. Al-Turjman and M. Abujubbeh, “IoT-enabled smart grid via SM: An overview,” *Future Generation Computer Systems*, vol. 96, pp. 579–590, Jul. 2019, doi: 10.1016/j.future.2019.02.012.

[4] B. Valks, M. H. Arkesteijn, A. Koutamanis, and A. C. den Heijer, “Towards a smart campus: supporting campus decisions with internet of things applications,” *Building Research and Information*, vol. 49, no. 1, pp. 1–20, Jan. 2020, doi: 10.1080/09613218.2020.1784702.

[5] H. Liu, “Smart campus student management system based on 5G network and internet of things,” *Microprocessors and Microsystems*, p. 103428, Nov. 2020, doi: 10.1016/j.micpro.2020.103428.

[6] K. Pal and A. U. H. Yasar, “Internet of things and blockchain technology in apparel manufacturing supply chain data management,” *Procedia Computer Science*, vol. 170, pp. 450–457, 2020, doi: 10.1016/j.procs.2020.03.088.

[7] A. Rejeb, S. Simske, K. Rejeb, H. Treiblmaier, and S. Zailani, “Internet of things research in supply chain management and logistics: A bibliometric analysis,” *Internet of Things (Netherlands)*, vol. 12, p. 100318, Dec. 2020, doi: 10.1016/j.iot.2020.100318.

[8] F. A. Alaba, M. Othman, I. A. T. Hashem, and F. Alotaibi, “Internet of things security: A survey,” *Journal of Network and Computer Applications*, vol. 88, pp. 10–28, Jun. 2017, doi: 10.1016/j.jnca.2017.04.002.

[9] M. Stoyanova, Y. Nikoloudakis, S. Panagiotakis, E. Pallis, and E. K. Markakis, “A survey on the internet of things (IoT) forensics: Challenges, approaches, and open issues,” *IEEE Communications Surveys and Tutorials*, vol. 22, no. 2, pp. 1191–1221, 2020, doi: 10.1109/COMST.2019.2962586.

[10] K. Tange, M. De Donno, X. Fafoutis, and N. Dragoni, “A Systematic survey of industrial internet of things security: Requirements and fog computing opportunities,” *IEEE Communications Surveys and Tutorials*, vol. 22, no. 4, pp. 2489–2520, 2020, doi: 10.1109/COMST.2020.3011208.

[11] A. V. Dastjerdi and R. Buyya, “Fog computing: helping the internet of things realize its potential,” *Computer*, vol. 49, no. 8, pp. 112–116, Aug. 2016, doi: 10.1109/MC.2016.245.

[12] R. O. Aburukba, M. AliKarrar, T. Landolsi, and K. El-Fakih, “Scheduling internet of things requests to minimize latency in hybrid fog–cloud computing,” *Future Generation Computer Systems*, vol. 111, pp. 539–551, Oct. 2020, doi: 10.1016/j.future.2019.09.039.

[13] M. R. Alizadeh, V. Khajehvand, A. M. Rahmani, and E. Akbari, “Task scheduling approaches in fog computing: A systematic review,” *International Journal of Communication Systems*, vol. 33, no. 16, p. e4583, Aug. 2020, doi: 10.1002/dac.4583.

[14] X. Yang and N. Rahmani, “Task scheduling mechanisms in fog computing: review, trends, and perspectives,” *Kybernetes*, vol. 50, no. 1, pp. 22–38, Mar. 2021, doi: 10.1108/K-10-2019-0666.

[15] K. Matrouk and K. Alatoun, “Scheduling algorithms in fog computing: A survey,” *International Journal of Networked and Distributed Computing*, vol. 9, no. 1, pp. 59–74, 2021, doi: 10.2991/IJNDC.K.210111.001.

[16] A. Hazra, M. Adhikari, T. Amgoth, and S. N. Srirama, “Joint computation offloading and scheduling optimization of IoT applications in fog networks,” *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 3266–3278, Oct. 2020, doi: 10.1109/TNSE.2020.3021792.

[17] Y. Yang, K. Wang, G. Zhang, X. Chen, X. Luo, and M. T. Zhou, “MEETS: maximal energy efficient task scheduling in homogeneous fog networks,” *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 4076–4087, Oct. 2018, doi: 10.1109/JIOT.2018.2846644.

[18] D. P. Abreu et al., “A rank scheduling mechanism for fog environments,” in *Proceedings - 2018 IEEE 6th International Conference on Future Internet of Things and Cloud, FiCloud 2018*, Aug. 2018, pp. 363–369, doi: 10.1109/FiCloud.2018.00059.

[19] Z. Liu, X. Yang, Y. Yang, K. Wang, and G. Mao, “DATS: Dispersive stable task scheduling in heterogeneous fog networks,” *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3423–3436, Apr. 2019, doi: 10.1109/JIOT.2018.2884720.

[20] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, “On QoS-aware scheduling of data stream applications over fog computing infrastructures,” in *Proceedings - IEEE Symposium on Computers and Communications*, Jul. 2016, vol. 2016-February, pp. 271–276, doi: 10.1109/ISCC.2015.7405527.

[21] M. Verma, N. Bhardwaj, and A. K. Yadav, “Real time efficient scheduling algorithm for load balancing in fog computing environment,” *International Journal of Information Technology and Computer Science*, vol. 8, no. 4, pp. 1–10, Apr. 2016, doi: 10.5815/ijitcs.2016.04.01.

[22] D. Rahbari and M. Nickray, “Low-latency and energy-efficient scheduling in fog-based IoT applications,” *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 27, no. 2, pp. 1406–1427, Mar. 2019, doi: 10.3906/elk-1810-47.

[23] Y. Sun, T. Dang, and J. Zhou, “User scheduling and cluster formation in fog computing based radio access networks,” in *2016 IEEE International Conference on Ubiquitous Wireless Broadband, ICUWB 2016*, Oct. 2016, pp. 1–4, doi: 10.1109/ICUWB.2016.7790393.

[24] S. Kabirzadeh, D. Rahbari, and M. Nickray, “A hyper heuristic algorithm for scheduling of fog networks,” in *Conference of Open Innovation Association, FRUCT*, Nov. 2018, pp. 148–155, doi: 10.23919/FRUCT.2017.8250177.

[25] D. Rahbari and M. Nickray, “Scheduling of fog networks with optimized knapsack by symbiotic organisms search,” in *Conference of Open Innovation Association, FRUCT*, Nov. 2018, pp. 278–283, doi: 10.23919/FRUCT.2017.8250193.

[26] H. Y. Wu and C. R. Lee, “Energy efficient scheduling for heterogeneous fog computing architectures,” in *Proceedings - International Computer Software and Applications Conference*, Jul. 2018, vol. 1, pp. 555–560, doi: 10.1109/COMPSAC.2018.00085.

[27] Y. C. Chen, Y. C. Chang, C. H. Chen, Y. S. Lin, J. L. Chen, and Y. Y. Chang, “Cloud-fog computing for information-centric internet-of-things applications,” in *Proceedings of the 2017 IEEE International Conference on Applied System Innovation: Applied System Innovation for Modern Technology, ICASI 2017*, 2017, pp. 637–640, doi: 10.1109/ICASI.2017.7988506.

BIOGRAPHIES OF AUTHORS

J. Geetha is working as an Associate Professorin Computer Science and Engineering Department of Ramaiah Institute of Technology, Bangalore. Her areas of interest include cloud computing, big data, semantic web, graph theory and web design. She has 17 years of teaching experience and has 21 publications, the most recent one being "Implementation and Performance Comparison of Partitioning Techniques in Apache Spark". Dr. Geetha is also a member of IEEE and ISTE. She can be contacted at email: geetha@msrit.edu.

Shaguftha Zuveria Kottur is a student who completed her B.E. at MSRIT Bangalore. She is currently pursuing M.E at IIIT Delhi and is working as a teaching assistant for an undergraduate algorithms course. Her areas of interest include algorithms, data plane programming, SDN and network security. She can be contacted at email: shagufthazk98@gmail.com.

Riya Ganiga is working at VMware as a Kubernetes Application Developer. She has expertise in various containerization platforms like Kubernetes, Openshift and Docker. She is proficient in networks, unix, operating systems and cloud platforms. She has a patent to herself from VMware pending USPTO (United States Patent and Trademark Office) approval. She has completed her B.E. at MSRIT Bangalore. She can be contacted at email: riyaganiga25@gmail.com.

D. S. Jayalakshmi is working as an Associate Professor in Computer Science and Engineering Department of Ramaiah Institute of Technology, Bangalore. She became a member of IAENG in 2020, member number: 139594. Her areas of interest include cloud computing, big data, computer graphics. She has 27 years of teaching experience and has 30 publications, the most recent one being "Simulation of MapReduce Across Geographically Distributed Datacentres Using CloudSim". Dr. Jayalakshmi is also a life member at ISTE. She can be contacted at email: jayalakshmids@msrit.edu.

Tallapalli Surabhi is working at Amazon as a Software Development Engineer - II for Automating Infrastructure of Amazon Go. Proficient in various programming languages and great team player. Her areas of interest include backend programming, algorithms. She has completed her B.E. at MSRIT Bangalore. She can be contacted at email: tallapallisurabhi15@gmail.com.