A Queuing Based Technique for Efficient Task Scheduling in Fog

Amrut Patro¹, Ranjit Kumar Behera²
¹²Department of Computer Science, National Institute of Science and Technology, Berhampur, India

e-mail: ¹patroamrut@gmail.com, ²ranjit.behera@gmail.com

Abstract. In recent years, with the rapid growth of the IoT devices, Fog computing also known as Edge computing has gained a lot of attention and is emerging as a popular computing technology for performing the IoT applications which requires immediate response in real-time. Fog as a virtualized layer of distributed network present between the Cloud and the end-users aims to process a large amount of data locally and send only the necessary information to the Cloud with a goal of reducing delay and cost. Fog having a limited amount of resources requires an efficient and systematic approach for scheduling of tasks. The sequence of scheduling the resources plays a crucial role in satisfying the deadlines of the requests. Hence, a Queuing based Technique (QBT) for efficient task scheduling in Fog has been proposed in this paper to support the rapidly increasing amount of data, enhance the performance and decrease the response time. Regarding this, an overall architecture is presented along with the flow of activities. Then corresponding node reservation and queuing mechanism with the algorithms are described. Experimental evaluation and results comparison is conducted to test the efficiency of the proposed QBT model.

1. INTRODUCTION

Due to the ever-increasing reliance on smart and intelligent applications, the technological innovations for efficient task management and processing in both Fog and Cloud computing are also taking place rapidly. Day by day the number of devices and the amount of data generated by those is also increasing enormously. Cisco’s Global Cloud Index estimates that around 850 Zettabytes of data will be generated by the IoT devices and the users by 2021 [1]. Cloud has been the most widely used computing technology since its inception [2, 3]. Not only this, for almost every IoT application cloud computing was used primarily for storage and computation [4]. Despite so many facilities provided by the cloud, some issues and drawbacks have been notices in Cloud which cannot be ignored. Cloud being geographically far-away from the end-users, leads to an intolerable increase in the delay to provide the service and this causes huge failures in completing the execution of tasks within their deadlines. Hence Fog was introduced by CISCO to overcome such drawbacks particularly for IoT based applications [5].

Fog computing, also known as Edge computing extends the Cloud computing capabilities to the edge of the network [6]. It is a virtualized layer of distributed network environment present between the Cloud and the end-users [7]. Fog is also based on virtualization which provides the Cloud services being close to the source of data. Fog being close to the IoT devices results in a reduced delay. The
need for Fog computing is realized more when a large number of delay-sensitive requests arrive which needs to be executed within their specific deadlines. Fog provides a supportive layer for the Cloud which processes a large amount of data locally and sends only the necessary information to the Cloud intending to reduce delay [8].

Though Fog looks to be more efficient than Cloud for delay-sensitive applications, due to the resource constraint in Fog, Cloud cannot be replaced altogether. Hence in most cases, both Cloud and Fog are used together to provide the best possible service to the end-users [9, 10]. In some situations when requests which require larger computing resources arrive which cannot be handled by the Fog alone, those requests are forwarded to the Cloud for processing. Fog having a limited amount of resources requires an efficient and systematic approach to schedule the tasks. An improper task scheduling may result in an increase in the cost and degradation of QoE leading to client dissatisfaction.

Recently there has been an increase in the number of studies on Fog computing with a goal of mitigating those drawbacks of Fog. In [11], authors have proposed a Delay and Energy Balanced Task Scheduling mechanism to achieve minimum service delay and less energy consumption. Whereas in [12], the authors have also proposed a Maximal Energy Efficient Task Scheduling model for homogeneous fog network similar to the previous work with the following exceptions. The authors have considered a Time Division Multiple Access (TDMA) system where, time-slot based task execution takes place. Though their methodology proved to be an energy efficient model, however some gaps can be noticed in their paper. Whenever a task arrives from a neighbor node, the helper nodes starts executing tasks from the neighbour task node which would result in an increase in the waiting time or service delay for the tasks present in its own task queue.

The authors in [13], have proposed a Dynamic Resource Allocation Method (DRAM), where they have tried to achieve maximum utilization of resources in the fog nodes. Their proposed algorithm consists of two methods, Static Resource Allocation, and Dynamic Service Migration. Using Static Resource Allocation method, they are first allocating tasks on fog nodes according to the descending order of the tasks resource requirements, and using Dynamic Service Migration, they are migrating tasks to a suitable fog node with a goal of achieving a better load balance variance. Their method proved to be successful in achieving high resource utilization in fog nodes however suffers from few drawbacks such as (i) Allocating tasks statically and again migrating them to fog nodes would result in an extra cost (migration cost) which is not considered in the paper. (ii) Also, the arrival of delay-sensitive tasks and task deadlines are not considered in the paper. (iii) Migration of partially executed tasks to other fog nodes would not prove to be a good idea as it causes unnecessary delay in execution.

The authors in [14], have proposed a prioritized task scheduling mechanism, where they have considered three priority levels as high, medium, and low for each task coming to the Fog for service. The delay tolerance level is checked according to the threshold values and the priority values are assigned to a request based on the factors such as subscription catalogue, delay tolerance level, service time, availability of resources in the fog layer and number of tasks already present in priority queues and thereafter the requests are placed into the respective priority queues. The requests are first forwarded to the nearest fog node for scheduling. If the nearest node is not able to schedule the request satisfying its deadline, then it is forwarded to other nearby servers for scheduling. If necessitates the task can also be divided into sub-tasks while forwarding to nearby fog servers to achieve minimum delay. In case the entire fog layer fails to schedule the task on any node, then it is forwarded to Cloud. To the best of our knowledge, this prioritized scheduling model has been the most efficient mechanism in achieving the minimum average response time for the tasks until now. But still some drawbacks have been found in their model. While sending the task to the nearby fog server and if the nearby fog server fails to schedule, then sending it to other nearby servers again, is not an efficient approach for delay-sensitive tasks as this process may increase service delay. Also in their model, it is found that some high priority tasks which are highly sensitive to delay are forwarded to Cloud, which will definitely increase the chances of failure in executing tasks within its deadline.

Hence keeping in mind the above challenges and issues we have proposed a model with a goal of achieving minimum average response time for the tasks and maximizing the number of tasks executed within their deadlines. So in this paper, we have proposed a queuing based technique (QBT) for
efficient task scheduling in the Fog environment, which is very much suitable for delay-sensitive tasks. Our major contributions are highlighted as follows:

We have proposed a ratio based node reservation technique, which reserves a certain amount of nodes for three priority levels (high, medium, and low) based on the ratio between tasks from each priority level with a goal of maximizing the chances of tasks getting resources.

A queuing based scheduling mechanism is proposed which mainly focuses on reducing the number of tasks getting forwarded to the Cloud.

Two algorithms are proposed for the aforementioned techniques and are simulated using CloudSim simulator. The comparison of our simulation results is done with the existing models shows that QBT achieves better performance in average response time. We have also shown how many tasks are forwarded to the cloud by assuming different Fog infrastructure.

The remainder of the paper is organized as follows. Section 2 describes the overall architecture which mainly describes the four layers considered in our paper. In Section 3 the flow of activities pertaining to the proposed model has been depicted. Section 4 presents the mathematical equations and algorithm design particularly the detailed description of Node Reserver and QBT models are given. Experimental set up and performance analysis is conducted in Section 5. In the end, the paper concludes with a summary in Section 6.

2. OVERALL ARCHITECTURE

The overall architecture which will be used for efficient task scheduling in Fog environment is depicted in fig.1. It consists of four layers namely Client, IoT interface, Fog Infrastructure, and Cloud. Clients are the end-users, can send their request for service to Fog infrastructure using IoT interface. IoT interface is a network of physical objects or devices having capabilities to communicate or exchange data over the internet such as laptop, smartwatch, smartphone, vehicle, etc. So, in our proposed architecture IoT interface acts as the medium by which the client is able to send requests to the Fog environment. Any request sent by the clients is first received by the Priority Assigner, present in the Fog Environment. The priority assigner will assign a priority value in terms of 3 levels namely high, medium, and low based on various parameters [14]. Thereafter these tasks are pushed into the respective priority queues from which Fog Controller will select the task for execution based on the priority levels.

As the number of tasks present in different priority queues will be different from time-to-time, to minimize the average response time a Node Reserver is used to a certain number of nodes for each priority level based on the ratio calculated. The Fog Controller plays a major role in scheduling the tasks on fog nodes. The Fog Controller selects tasks from the priority queues and then looks for a suitable node from the respective reserved priority nodes for scheduling. In case, there is insufficient nodes to accommodate all higher priority task in its reserved nodes, then Fog Controller can use
reserve nodes from lower priority levels instead of rejecting the task or forwarding it to the cloud. Forwarding the tasks to the cloud probably will not be a good idea as there is an uncertainty of delay in the cloud.

Hence, the use of such a strategy reduces the number of tasks forwarded to the cloud up to a great extent. Each Fog nodes in our proposed model consists of two local queues namely, ‘own-tasks-queue’ and ‘neighbor-tasks-queue’. Whenever a task of the same priority is scheduled on a fog node, it is pushed into the own task queue of the node whereas when a task from higher priority is assigned on a fog node, it is pushed into the neighbour task queue of the node. When there is high traffic and no suitable node is available for scheduling in the Fog environment then the requests are forwarded to the cloud.

3. FLOW OF ACTIVITIES

In this section, the flow of activities starting from the arrival of requests till the scheduling of those requests is presented. All the sequence of steps are clearly shown in Fig 2.

3.1. Sends request: This is the first step, where client sends request to the Fog via IoT interface. Each request arrives with its own deadline specified in the SLA.

3.2. Priority assigned and pushed into queue: Whenever a request comes into the fog environment, the Priority Assigner assigns a priority level (high, medium, or low) to the request and pushes it into the respective priority queue. The priority assigner assigns priority level to the tasks based on certain parameters like subscription catalogue, delay tolerance level and estimated service time of the task [14]. The three priority queues are then submitted to Node Reserver and Fog Controller for further scheduling.

3.3. Reservation of nodes takes place: The Node Reserver after receiving the three priority queues finds the ratio between total number of tasks present inside each priority queue and reserves fog nodes for each priority level according to the ratio calculated. The reservation of nodes for the three priority levels ensures minimum failure of tasks not getting scheduled on the fog nodes as certain amount of resources kept fixed for the respective priority tasks.

3.4. Priority queues received: The priority queues are received by the Fog Controller and thereafter the reservation of nodes is done by the Node Reserver. Once the node reservation is finished, the scheduling of a task by the Fog Controller can be performed by the following three steps as follows.

   a) Scheduling of task on a suitable node of same priority level: Whenever a task is selected from any of the priority queues the Fog Controller at first does some checks and tries to schedule the task (pushed into the own task queue) on a fog node with same priority. A suitable node for a task refers to a node whose expected delay for scheduling the task does not exceed the maximum tolerable delay of that particular task. In case a node is heavily loaded, and cannot execute the particular task within its deadline then it is not considered as a suitable node.

![Figure 2. Flow of activities](image-url)
b) In case (a) is failed, then scheduling is done on a suitable node selected from lower priority level. In a situation when there is no suitable node with same priority is found the Fog Controller checks if there is any suitable node found with lower priority levels. If available then the Fog Controller schedules the task on the fog node (pushed into the neighbour task queue). Whenever such situation occurs in which a higher priority task is pushed into the neighbour task queue of a lower priority node, the fog nodes first executes the tasks from neighbour task queue then resumes executing tasks from their own task queue. This is done because tasks in neighbour task queue have less delay tolerance and needs to be executed as soon as possible without any delay.

c) In case both (a) and (b) failed, then the task will be forwarded to the cloud: This can be considered as the worst case scenario when there is a high traffic and no suitable node is found in the entire fog environment which can satisfy the deadlines of the request, then in such a case, the request is forwarded to the cloud.

4. MATHEMATICAL EQUATIONS AND ALGORITHMS

In the fog environment various requests arriving from the client may have different tolerable delays and these requests need to be served within their deadlines. Implementation of various concepts and calculations like ratio between priority tasks, expected delay calculation, delay due to queue backlog etc. are described in this section. The notations used in our model is presented in Table 1.

4.1. Node Reserver

The equation assumptions and the algorithm design used for reservation of nodes are presented here.

Let ratioH : ratioM : ratioL be the prime ratio between lenQueH, lenQueM and lenQueL.

Then the total number of parts of the ratio can be represented as,

\[ \text{totalRatParts} = \text{ratioH} + \text{ratioM} + \text{ratioL} \]  

(1)

Dividing the amount by the total number of fog nodes present, each part of the ratio can be calculated,

\[ \text{eachRatPart} = \frac{\text{totalF}}{\text{totalRatParts}} \]  

(2)

Now, the number of fog nodes which needs to reserved for high, medium and low priority tasks respectively are

\[ \text{reserveH} = \text{ratioH} \times \text{eachRatPart} \]  

(3)

\[ \text{reserveM} = \text{ratioM} \times \text{eachRatPart} \]  

(4)

\[ \text{reserveL} = \text{ratioL} \times \text{eachRatPart} \]  

(5)

Hence, the ratio expressed as

\[ \text{reserveH} : \text{reserveM} : \text{reserveL} \]  

(6)

Table 1. Notation Table

| Notation        | Description                                                                 |
|-----------------|-----------------------------------------------------------------------------|
| queueH, queueM, queueL | Priority queues of level high, medium and low                                |
| priorityL\textsubscript{i} | Priority level of \textit{i}\textsuperscript{th} request                      |
| own\textsuperscript{\textsubscript{TQ}}\textsubscript{n} | The own task queue of \textit{n}\textsuperscript{th} fog node                |
| neighbour\textsuperscript{\textsubscript{TQ}}\textsubscript{n} | The neighbour task queue of \textit{n}\textsuperscript{th} fog node          |
| lenNeigh\textsuperscript{\textsubscript{TQ}}\textsubscript{n} | Total number of tasks present in neighbour task queue of \textit{n}\textsuperscript{th} fog node |
| lenOwn\textsuperscript{\textsubscript{TQ}}\textsubscript{n} | Total number of tasks present in own task queue of \textit{n}\textsuperscript{th} fog node |
| max\textsuperscript{\textsubscript{Delay}}\textsubscript{i} | Maximum delay which can be tolerated by \textit{i}\textsuperscript{th} request |
| exp\textsuperscript{\textsubscript{Delay}}\textsubscript{\textsubscript{\textsubscript{\textsubscript{\textsuperscript{T}}}\textsuperscript{\textsubscript{n}}}}\textsubscript{(t)} | Expected delay on \textit{n}\textsuperscript{th} fog node at time instant \textit{t} |
| exp\textsuperscript{\textsubscript{Delay}}\textsubscript{\textsubscript{\textsuperscript{\textsuperscript{i}}\textsuperscript{(t)}}} | Expected delay for \textit{i}\textsuperscript{th} request on \textit{n}\textsuperscript{th} fog node |
| num\textsuperscript{\textsubscript{TE\textsuperscript{\textsubscript{exec}}}}\textsubscript{\textsubscript{\textsubscript{\textsuperscript{\textsuperscript{\textsuperscript{T}}}\textsuperscript{\textsuperscript{n}}}}\textsubscript{(t)}} | Number of tasks executing in \textit{n}\textsuperscript{th} fog node at time instant \textit{t} |
| exp\textsuperscript{\textsubscript{Remain\textsuperscript{\textsubscript{ST}}}\textsubscript{\textsubscript{\textsubscript{\textsuperscript{\textsuperscript{\textsuperscript{i}}}\textsuperscript{(t)}}}}} | Expected remaining service time of \textit{i}\textsuperscript{th} request currently |
executing on $n^{th}$ fog node at time instant $t$

$\text{expTotalSTQ}_H, \text{expTotalSTQ}_M, \text{expTotalSTQ}_L$

$\text{Expected total service time of tasks present in high, medium and low priority queues}$

$\text{waitTQ}_i$

$\text{Time spent by } i^{th} \text{ request in priority queue}$

Following the equations and the model, Node Reserver Algorithm is presented which reserves nodes for the three priority levels based on the ratio between the total number of requests in the three priority queues. At first its high-level algorithm is presented, followed by a detailed explanation of how the algorithm works.

**Algorithm 1 (Node Reserver Algorithm)** depicts the behaviour of a Node Reserver whose main goal is to reserve certain number of fog nodes for the three priority level tasks. A task at first is tried to be scheduled on the own task queue of a reserve node of the same priority. The algorithm first calculates the prime ratio between the $\text{lenQueH}, \text{lenQueM}$ and $\text{lenQueL}$ (line (1)). Then the number of nodes need to be reserved for each priority level is calculated based on the prime ratio found (line (2-4)). The list of fog nodes are traversed and at first a certain number of nodes which was calculated to be reserved for high priority tasks are kept reserved for them and then the reservation of nodes for medium and low priority tasks is done (line (5-17)). The goal of Node Reserver Algorithm is to reserve a certain amount of resources in terms of number of nodes for each priority level so that enough resources will be kept reserved for the tasks of respective priority levels and there would be a less chances of tasks not getting a suitable node for scheduling.

Algorithm 1. Node Reserver Algorithm

| Algorithm 1. Node Reserver Algorithm |
|--------------------------------------|
| **Input:** $\text{lenQueH, lenQueM, lenQueL}$ |
| **Output:** Scheduling of requestable tasks |
| 1. $\text{Calculate ratioH = ratioM = ratioL}$ |
| 2. $\text{Calculate totalNumParts} = \text{lenQueH/totalNumParts}$ |
| 3. $\text{calculatePartH=totalNumParts/totalNumParts}$ |
| 4. $\text{Calculate reservedH = reservedM = reservedL}$ |
| *using equations (1-3)* |
| 5. $\text{countH} = \text{countH} = \text{countH} = 0$ |
| 6. for $\text{partH} in \text{listH}$ |
| 7. if $\text{countH} = \text{reservedH}$ then |
| 8. priorityH = High |
| 9. $\text{countH} = \text{countH} + 1$ |
| 10. else if $\text{countH} = \text{reservedM}$ then |
| 11. priorityM = Medium |
| 12. $\text{countH} = \text{countH} + 1$ |
| 13. else if $\text{countH} = \text{reservedL}$ then |
| 14. priorityL = Low |
| 15. $\text{countH} = \text{countH} + 1$ |
| 16. end if |
| 17. end for |

4.2. Proposed QBT

The proposed QBT algorithm plays the main role in scheduling tasks on fog nodes. The assumptions and derivation of various equations used for implementing the QBT are presented in this section followed by the algorithm design and its description.

The expect total service time of tasks present in high priority queue can be found as,

$$\text{expTotalSTQ}_H = \sum_{i=1}^{\text{lenQueH}} \text{expST}_i$$  \hspace{1cm} (7)

where, $i =$ the $i^{th}$ request present in queueH. Similarly,
can also be calculated. The maximum tolerable delay for the ith request is 
\[
\text{maxDelay}_i = \text{deadLine}_i - 
\text{currentT}
\]

where, \(\text{deadLine}_i\) = Deadline of ith request as specified in the SLA. 
\(\text{currentT}\) = The time instant at which maxDelay\(_i\) is being calculated.

The expected delay by nth fog node at time instant t is calculated as
\[
\text{expDelay}_{Fn}(t) = \text{waitTQ}_i + \text{expST}_i + \text{expDelay}_{Fn}(t)
\]

And a task can be scheduled on a node if, 
\[
\text{expDelay}_{Fn}(t) < \text{maxDelay}_i
\]

The Algorithm2 presented further implements the actual QBT mechanism for efficiently scheduling the tasks on fog nodes. A detailed discussion on how the QBT algorithm works is done at the end of this section.

Algorithm2 (QBT Algorithm) is designed to implement the QBT mechanism which is used by the Fog Controller to schedule the tasks on fog nodes. The Fog Controller continues scheduling of tasks till there are requests present in the three priority queue (line (1)). For each request in the priority queues the maximum tolerable delay is first calculated (line (2)) and the total expected delay to be faced if scheduling is done on the node of same priority is checked (line (4,23,32)). If expected delay for scheduling on nodes with same priority is not more than the maximum tolerable delay of the request then a suitable node of same priority is found for scheduling (line (14- 21, 29, 35)). In case if expected delay for scheduling on nodes with same priority is found to be greater than the maximum tolerable delay then a suitable node is searched having priority lower then the actual priority of the task and is scheduled into the neighbour task queue of the suitable node (line (5-10, 24)). The fog nodes always executes the tasks from neighbour task queue rather than executing the tasks from own task queue. This is because mostly the tasks in neighbour queue have less tolerance capacity than the tasks in own task queue as the neighbour tasks are always from a higher priority level. 

In case of a high priority task if delay for scheduling the task on high priority nodes is found to be greater than the maximum tolerable delay of the task then suitable node with low priority level is first searched for scheduling, if not available then a suitable node is searched with medium priority (line (5)). However, in the same case for a low priority task if expected delay for scheduling on same priority nodes is found to be greater than the maximum tolerable delay then it is directly forwarded to the cloud (line (33)). In worst case scenarios, if there are no suitable nodes found in the entire fog environment for high, medium priority tasks, the request is forwarded to the cloud (line (11-13, 25-27)). The main focus of the QBT Algorithm is to schedule tasks on fog nodes in such a way that the deadlines of the requests are satisfied and it also focuses on reducing the number of tasks forwarded to cloud.
5. EXPERIMENTAL SETUP AND RESULT ANALYSIS

In this section, we have given the detail description regarding the experimental setup and evaluation of our proposed model. The description about input, output and various parameters considered for the simulation purpose is given in the subsection 5.1, followed by the analysis of the results achieved by simulation in subsection 5.2.

5.1. Experimental Setup

For the simulation and analysis of the proposed model, we have used CloudSim simulator [15]. Three different Cloudlet lists for the three priority levels were used. Each cloudlet list consists of tasks with having priority. Two different datacenters CloudDC and FogDC with specifications such as x86 system architecture, Xen vmm and Linux operating system were used for depicting the behaviour of Fog and Cloud infrastructure. The three Cloudlet lists were submitted to three different instance of Broker. The simulator was setup to periodically generate a set of requests and push them into the respected Cloudlet list based on the task priority. For defining a fog infrastructure with ‘n’ number of fog nodes, ‘n’ number of hosts were defined in the datacenter FogDC with each. The various parameters considered during the evaluation are presented in Table 2.

Table 2. Evaluation Parameters Table

| Parameter                        | Domain                  |
|----------------------------------|-------------------------|
| Number of tasks                  | [10 to 100]             |
| Number of fog nodes              | [10,50]                 |
| Expected service time of tasks (secs) | [0.02,0.03,0.04]         |
| Maximum tolerable delay of tasks (secs) | [0.05,0.2,0.4]         |
| Number of tasks executing on a fog node at time instance t | [1 to 3] |

5.2. Performance Evaluation

We have evaluated the performance of our proposed method by considering mainly two parameters; one is average response time for the incoming tasks and another is number tasks forwarded to the cloud and we compare our result with prioritized algorithm [14].

5.2.1. Performance evaluation on average response time: For the evaluation of average response time, we have considered two different Fog infrastructure; one with a relatively small value i.e. 10 number of node and another with a high value i.e. 50 number of nodes. For both the cases we have calculated the average response time by assuming variable incoming tasks from 10 to 100. Fig. 3 depicted the result with 10 Fog nodes whereas, Fig. 4 depicted the result with 50 nodes. From both the Fig.3 & Fig.4, it is clearly visible that our proposed QBT model outperformed the Prioritized technique in terms of average response time. Generally, for scenarios when the number of tasks is less than the number of nodes, the response time should remain unchanged till the incoming number of tasks is not greater than the number of nodes.
However, from Fig. 4, it is noticeable that up to 40 tasks there was a minimal increase in the average response time, which is because of the delay caused while traversing all the fog nodes of same level and checking its queue backlog for finding a suitable node.

The average response time for ‘N’ number of tasks can be calculated as

$$\text{ResponseT} = \frac{1}{N} \sum_{i=1}^{N} (\text{ExpectedRT}_i + \Delta T)$$  \hspace{1cm} (14)

where, \( \text{ResponseT}_i \) is the response time for \( i^{th} \) task,

And, \( \Delta T \) is the negligible delay which can be calculated as

$$\Delta T = \sum_{i=1}^{\text{TotalFN}} (\text{LoopDelay}_{i} + \text{QTraverseDelay})$$  \hspace{1cm} (15)

where,

- \( \text{TotalFN} \) is the total number of fog nodes of same level,
- \( \text{LoopDelay} \) is the delay for traversing the fog nodes,
- \( \text{QTraverseDelay} \) is the delay for traversing and checking queue backlog of \( i^{th} \) fog node.

But in later stages with an increase in number of tasks it is found that the proposed QBT model performs better than the existing prioritized model and is true for both the Fog infrastructure.

5.2.2. Performance evaluation on number of tasks forwarded to Cloud: The number of tasks forwarded to Cloud plays a very important role in deciding the number of tasks getting failed to be executed within their deadlines. Forwarding tasks to Cloud would prove to be disastrous for the delay sensitive or the high priority tasks due to an uncertainty of delays in the Cloud. This also leads to the degradation of QoE. To the best of our knowledge, there are a very few studies which have focused on minimizing the number of tasks forwarded to Cloud. Hence, keeping this issue in our mind, we have tried to design our model with a goal of minimizing the number of tasks forwarded to Cloud.

![Figure 5. Number of tasks forwarded to Cloud when 10 and 50 Fog nodes are considered](image)

Fig. 5 shows the number of tasks forwarded to Cloud considering Fog infrastructure of 10 and 50 nodes respectively, with increasing number of tasks. More number of tasks in fog means more traffic. With increase in the traffic the availability of resources generally decreases in fog which in return results in decreasing the chances of tasks getting scheduled in Fog and increasing the number of tasks getting forwarded to Cloud. As we can see from the bar graph depicted in Fig. 5 that when less number of tasks arrived the number of tasks forwarded to Cloud is almost nil. Even when we consider a large Fog infrastructure of 50 nodes, which is 5 times greater than the first infrastructure, the gap is quite low. Hence, we conclude that our proposed model is very efficient to handle almost every incoming task within their deadline even with a small number of nodes.
6. CONCLUSION

The explosive increase in the number of IoT devices has resulted in an increase in the data generated by these devices. Fog computing, which extends the Cloud computing capabilities to the edge of the network seems to be efficient in handling delay-sensitive applications. But due to the resource constraint in Fog, arriving tasks may suffer from lack of resources and delays in getting responses. Henceforth, efficient scheduling of tasks can further minimize the service delays and can increase the performance. Being mindful of these facts, in this paper, we have proposed a queuing based technique for efficient scheduling in Fog with a goal of providing immediate response to the delay sensitive tasks as well as executing the tasks within their specific deadlines. Through result analysis, it is undeniable that the proposed QBT model outperforms the existing model like prioritized scheduling to a great extent. In the future, we shall investigate the task scheduling algorithm by applying load balancing and energy-saving strategies, along with a focus on maximizing the resource utilization of fog nodes.

REFERENCES

[1] https://www.cisco.com/c/en/us/solutions/service-provider/a-network-to-support-iot.html [Accessed on 12/07/2020 at 2:00 PM]
[2] Armbrust, Michael, et al. "A view of cloud computing." Communications of the ACM 53.4 (2010): 50-58.
[3] Brian, Hayes, et al. "Cloud computing." Communications of the ACM 51.7 (2008): 9-11.
[4] Kaur, Maninder Jeet, and Piyush Maheshwari. "Building smart cities applications using IoT and cloud-based architectures." 2016 International Conference on Industrial Informatics and Computer Systems (CIICS). IEEE, 2016
[5] https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf [Accessed on 12/07/2020 at 2:00 PM]
[6] Yi, Shanhe, et al. "Fog computing: Platform and applications." 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb). IEEE, 2015.
[7] Naranjo, Paola G. Vinueza, et al. "FOCAN: A Fog-supported smart city network architecture for management of applications in the Internet of Everything environments." Journal of Parallel and Distributed Computing 132 (2019): 274-283.
[8] Khattak, Hasan Ali, et al. "Utilization and load balancing in fog servers for health applications." EURASIP Journal on Wireless Communications and Networking 2019.1 (2019): 91.
[9] Roy, Diptendu Sinha, et al. "A context-aware fog enabled scheme for real-time cross-vertical IoT applications." IEEE Internet of Things Journal 6.2 (2018): 2400-2412.
[10] Reddy, K. Hemant Kumar, et al. "A Service Delay Minimization Scheme for QoS Constrained, Context Aware Unified IoT Applications." IEEE Internet of Things Journal (2020).
[11] Yang, Yang, et al. "DEBTS: Delay energy balanced task scheduling in homogeneous fog networks." IEEE Internet of Things Journal 5.3 (2018): 2094-2106.
[12] Yang, Yang, et al. "MEETS: Maximal energy efficient task scheduling in homogeneous fog networks." IEEE Internet of Things Journal 5.5 (2018): 4076-4087.
[13] Xu, Xiaolong, et al. "Dynamic resource allocation for load balancing in fog environment." Wireless Communications and Mobile Computing 2018 (2018).
[14] Choudhari, Tejaswini, Melody Moh, and Teng-Sheng Moh. "Prioritized task scheduling in fog computing." Proceedings of the ACMSE 2018 Conference. 2018.
[15] Calheiros, Rodrigo N., et al. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms." Software: Practice and experience 41.1 (2011): 23-50.