Computers, Materials & Continua
DOI:10.32604/cmc.2021.014321
Article

Tech Science Press

State-Based Offloading Model for Improving Response Rate of IoT Services

K. Sakthidasan[1], Bhekisipho Twala[2], S. Yuvaraj[3], K. Vijayan[3,*], S. Praveenkumar[3], Prashant Mani[4] and C. Bharatiraja[3]

[1]Department of Electronics and Communication Engineering, Hindustan Institute of Technology and Science, Chennai, 603103, India
[2]Faculty of Engineering Built Environment, Durban University of Technology (DUT) University, Durban, 4001, South Africa
[3]SRM Institute of Science and Technology, Kattankulathur, 603203, India
[4]SRM Institute of Science and Technology, NCR Campus, Modinagar, 201204, India
[*]Corresponding Author: K. Vijayan. Email: vijayank@srmist.edu.in
Received: 13 September 2020; Accepted: 06 November 2020

Abstract: The Internet of Things (IoT) is a heterogeneous information sharing and access platform that provides services in a pervasive manner. Task and computation offloading in the IoT helps to improve the response rate and the availability of resources. Task offloading in a service-centric IoT environment mitigates the complexity in response delivery and request processing. In this paper, the state-based task offloading method (STOM) is introduced with a view to maximize the service response rate and reduce the response time of the varying request densities. The proposed method is designed using the Markov decision-making model to improve the rate of requests processed. By defining optimal states and filtering the actions based on the probability of response and request analysis, this method achieves less response time. Based on the defined states, request processing and resource allocations are performed to reduce the backlogs in handling multiple requests. The proposed method is verified for the response rate and time for the varying requests and processing servers through an experimental analysis. From the experimental analysis, the proposed method is found to improve response rate and reduce backlogs, response time, and offloading factor by 11.5%, 20.19%, 20.31%, and 8.85%, respectively.

Keywords: Decision-making; internet of things; markov model; task offloading; task graph

1 Introduction

The Internet of Things (IoT) is one of the technologies that interconnect many devices for data transmission over a network. The data is transferred without any human-to-human or human-to-computer communication [1]. IoT interacts with heterogeneous devices to improve data quality and for visualization of the applications and services [2]. This platform is capable

of providing pervasive access to resources and services and improving the response rate of the application. The heterogeneous platform is composed of tiny sensors, distributed networks, cloud, fog and edge computing paradigms, and human-computer interactions, among others. Besides, the computing paradigms include various distributed and parallel systems and processing elements for faster responses [3,4]. IoT offers scalable and flexible service responses for the users, irrespective of the requests. The density of requests and the available resources are efficiently managed by the IoT devices and cloud infrastructures for providing an effective user experience [5,6]. Task offloading is the process of transferring the task to the service provider through the external platform which includes the cloud or a grid. IoT task offloading is used in fog computing to increase the congestion in the network and decreases the response time [7]. The task offloading is having the capabilities to access the end device of the resource. In this case, the task offloads the resources to the intermediate platform. From the intermediary, the task is offloaded to the cloud environment [8].

The computation offloading is a centralized service provider that handles a set of instructions for input and output tasks. It is used in the remote server in the open cloud [9]. The device that is having computational offloading is ignored by its workload. It gives better performance and resource transfer [10,11]. Computational offloading sent the heavy resource to the servers and in return, it gets the results from the servers. Server is a service requestor and service provider [12]. It varies from the migration model which is used as multi-processor and grid computing. The offloading is done by using the decision process, which task is given to the service provider [13]. By making decisions the service response time is enhanced and also it reduces the time delay [14]. A reliable computation and task offloading reduce the complexity and processing steps in handling user requests. The scalable and flexible nature of the IoT platform acquires multi-level requests from different applications as service requirements [15,16]. The heterogeneous requests are to be responded with different services without backlogs or response losses. IoT incorporates different offloading techniques for improving the response rate for the users [17]. This is achieved by reducing the offloading factor, backlogs and responding time [18].

The contributions of the paper are as follows:

- Designing a state-based task offloading method (STOM) for improving the reliability of IoT services response irrespective of the request density.
- Introducing a directed task graph for identifying the states of the resources and their acceptance rate of improving the service response and reducing response time.
- Designing a task offloading process for verifying the states of the resources to reduce the backlogs and offloading requirements.
- Performing a comparative analysis for verifying the consistency of the proposed STOM using the metrics response rate, response time, backlogs, and offloading factor.

The manuscript is organized as follows. Section 2, provides investigation related works in offloading model for improving response rate of IoT services explained and compared. Section 3, gives brief description and application of the state-based task offloading method. The experimentation and results were discussed in Section 4. The conclusion is given in Section 5.

2 Related Works

Nowadays many researchers have focused on task offloading and resource allocation strategies when dealing with the response time. Others have considered different scenarios such as user [single user or multiple users], base station, task computation [partial or full offloading] and

computation environments [dynamic or static]. Wireless technology generates important cloud computing applications like online gaming, augmented reality and real time data streaming. By setting up computation networks at user side and avoiding generated traffic from application in a distant data center, Mobile Edge Computing (MEC) accommodate a fruitful way to communicate between edge server and users. Eventually it can reduce delay while executing all computation tasks and this can also save the energy consumption while dealing with time sensitive application in cloud computing. In this, decision making is very important to analyse whether the wireless devices are getting offloaded at the edge server or not. When tasks are offloaded to the mobile edge server, a bottleneck may appear on the upward communication channels. This tends to a compelling delay in processing computation tasks. In this way, to misuse the computation offloading, it requires joint administration for offloading choice and the related radio bandwidth allocation. Because of two-fold property of binary offloading decision, straight forwardly specifies each one of those potential solutions in computationally restricted. The below Tab. 1 depicts the existing offloading framework carried out so far.

Table 1: Existing offloading framework

Framework	Cloud resource type	Code offloading	Context-aware execution monitoring	Task allocation & scheduling	Fault tolerance
Mobile FBP	Nearby servers	Flow-based programming	Resource profiling	Single user, heuristic	–
COMET	Public clouds	Distributed shared memory	Thread execution profiling	Single user, heuristic	Checkpoint
MAUI	Nearbyservers	.Net common language runtime	Resource profiling	Single user, minimum make span and energy, ILP	Checkpoint
Cuckoo	Nearby servers	–	Not required	Single user, heuristic	–
JADE	Public clouds	Mobile agent(Java)	Analytic models	Single user, heuristic	–
Think Air	Public clouds	Java reflection	Resource profiling	Single user, heuristic	Memory error detection
Spectra	Nearbyservers	–	–	Single user, heuristic	Proactive prediction on usage demand
Handoff scheme	Mobile device cloud and public cloud	–	Resource profiling and discovery	Multi users, heuristic	Fuzzy multi-criteria decision making based handover strategy
Follow-Me Cloud	Public clouds	–	Resource profiling	Single user, minimum make span, MDP	–

Sun et al. [19] formulated a computation offloading game to allocate fog nodes efficiently with minimal processing. Concerning an increase in QoE (quality of experience) by analyzing the reduction of computation energy and system response delay are calculated in IoT systems.

The Computation Offloading Game in IoT is proposed by Shah-Mansouri et al. [20] for hierarchical Fog-cloud computing. The work in [19] requires the cooperative participation of multiple service providers whereas in [20], this is mitigated. This work aims to maximize the user quality of experience (QoE). The computation of service is used to decreases energy and delay.

Different from the game based offloading in [20], Ning et al. [21], observed a Computation Offloading Scheme that is cooperative partial, is developed for mobile edge computing (MEC) in IoT. The proposed method has multi-user which express a Mixed Integer Linear Programming (MILP) between the resources. The drawback of high offloading factor is reduced in the proposal given in [21]. Computation offloading method (COM) is introduced for big data that allows cloud-edge computing for resolving multi-objective task issues that are addressed by Xu et al. [22]. This method increases the processing and energy consumption of the end device. The COM is reliable in mitigating the issue of the backlog in [19] and [21] due to the non-cooperative resource allocations.

Alam et al. [23], developed an automatic computation for IoT applications. It uses mobile edge offloading technique in a heterogeneity environment. A Q-learning method is proposed to enhance the performance and reduce the latency of computing of IoT. The problem with this proposal is the non-allocation of prolonged requests due to consecutive request processing. Therefore, the response time in this is quite high than the cooperative method in [20]. To overcome the response time issues in [20], adaptive offloading is done on a genetic algorithm for enhancing the IoT device messaging reliably was introduced by Hussain et al. [24]. The work identifies unwanted request which affects the data transmission rate. The gateway monitors the different types of necessary communication. Monitoring necessary communication requires more sophisticated components and hence the scalability of the system is not retained.

Kotb et al. [25], observed Multi-agent cooperation for IoT devices that are done by Workflow-nets in a cloud environment. The cooperative operator is used to obtain the topology that is resulting in the fog computing agents. The aim of this is to reduce the optimization problem. The update of service availability and request density for different time intervals leads to better response time. Even though the response time issues were resolved up to an extent in the above investigations using various algorithms and computations, the dynamic allocation of states to enhance the response time and reduction in delay found to be still lagging. The novelty of our proposed work is to design a state-based task offloading method (STOM) for improving the reliability & the identification of states were done using a directed task graph concerning the service response. Dynamic allocation of states was carried out to reduce the backlogs and offloading requirements. From the experimental results, verified the consistency of the proposed STOM using the metrics response rate, response time, backlogs, and offloading factor. The offloading decisions are made using the Markov model for admitting the requests without backlogs. The proposed method is found to achieve 11.5% high response rate, 20.31% less response time, 20.19% fewer backlogs, and 8.85% less an offloading factor.

3 State-Based Task Offloading Method (STOM)

An IoT task offloading in a cloud environment is performed through wireless communication medium and connected-infrastructure components. Task offloading is used to transfer the process from one service unit to another. Here, computational task offloading is used to exchange service requests and responses between different connected units. The computational time and task are considered using a Markov decision process. In this work, it addresses two objectives such as

- Maximizing the service response rate and
- Reducing time delay

This process requires a directed task graph definition and decision-making process that is discussed in the following sections.

3.1 Directed Task Graph

The process is defined in the directed task-oriented graph and is represented as $G(v,e)$. In this graph structure, the nodes are denoted as task v and the edges are denoted as task dependency e. The task offloading is represented as $t \in v$ the weight is assigned to the task w_v. A directed graph is structured as $(a,b) \in e$ where forwarding is done through the task a *to* b. Variable a is denoted as a parent task, b which is represented as a child task that has lesser weight. The weight is denoted as $w_v(a,b)$ that indicates the weight for every task. The processing time of the single task is calculated by using Eq. (1).

$$x = \frac{w_v}{p_t} * (a,b) \tag{1}$$

From the above Eq. (1), x is the time for evaluation of the task, p_t is denoted as processing the task, t is denoted as a task. The processing is calculated for every single task. The cloud provides the task to the user to make decisions concerning with offloading. The set of the task is queued in the server and the task is transferred by using data transmission formulated in Eq. (2).

$$d_t = s_t + c_o * \frac{u_t}{p_t} \tag{2}$$

Eq. (1) is used to find the processing time of the task and this estimates the transfer of task from a *to* b, d_t is denoted as data transmission, s_t states the size of the task. c_o Represents the computational offloading, u_t refers to the execution time of the task. The queue timing is used for offloading the task in the cloud, which transfers the tasks from parent to child task. The offloading task is used to reduce the time delay. The delay is found through calculating the time of transmitting the task from source to destination.

In this work the delay is observed by transfer of task from one to another by doing some delay in the task transfer. The task transfer process is done through the sequence of offloading. In the directed task graph $G(v,e)$, the pursuing task is assigned as 1, for the perusing task the time is calculated. The sequential task is calculated using this condition $t\,(n,r) < t_{max}$, where the task is given to b, and its processing time is calculated in the following Eq. (3).

$$x_t\,(n,r) = \sum_{a=0}^{b-1} b_t + m_t + \sum_{a=b}^{m} q_t + m_{t,t+1} + \sum_{a=m+1}^{m} x_{a,b} \tag{3}$$

From Eq. (2) the data transmission is calculated for the task assigned and from that the Eq. (3) is used for analysing the task in the sequential way which reduces the delay. In Eq. (3), n is the entry task, r is denoted as an exist task and m is denoted as task processing and q_t represents the queuing task. This equation is used to find the entry and exit task and the time is

calculated by finding its processing time. By using Eqs. (2) and (3) the following Eq. (4) is used for deriving the condition for delay and response time.

$$d_t(x_t) = \begin{cases} \dfrac{u_t}{p_t} + \left[\sum_{a=b}^{m} q_t + m_{t,t+1}\right]_{n,r} < L_{max} \\ \dfrac{u_t}{p_t} + \left[\sum_{a=b}^{m} q_t + m_{t,t+1}\right]_{n,r} > L_{max} \end{cases} \tag{4}$$

Eq. (4) is having two conditions the first one is $\dfrac{u_t}{p_t} + \left[\sum_{a=b}^{m} q_t + m_{t,t+1}\right]_{n,r} < L_{max}$ in this case, when the processing time and the execution time are summed with queuing tasks. Where L is the time taken for task processing. The entry and exit tasks are calculated by using n, r is lesser than the time which is fixed to the maximum range that means the delay is less and it increases the response rate also. In other cases $\dfrac{u_t}{p_t} + \left[\sum_{a=b}^{m} q_t + m_{t,t+1}\right]_{n,r} > L_{max}$ is having the larger processing time when compared to the before condition.

The graph is formed as the parent task is having the highest weight and the child task is assigned as lesser weight by calculating Eqs. (1) and (3). Using this only the tree structure is formed, it satisfies the condition when there is no task in the queue and it is represented as q_t is denoted as queuing task is equal to null $q_t = \varnothing$. Instead, this condition is true for the first case means the task is offloaded, if not the process is continued until the condition is satisfied.

Thus, the computational task offloading is done using the directed task graph representation model. The task is been represented using the tree-based structure using the task and dependency of the task. The parent and child task is distributed concerning the weights. The processing is done by making the decision. The IoT for task offloading is processed based on the cloud-based information. The entry task and the existing task are assigned by the service provider in the network. The task processing time is computed in a required period which is obtained in Eq. (4) as the condition. From this, the condition is done through the decision-making process which is done through the Markov Decision Process for analyzing the task state and action.

3.2 Markov Decision Process for Task Offloading

The Markov decision process is used to maximize the response rate and also reduces the delay. This includes the state and action for the task. In this Markov decision process, it takes a finite set of states and actions. Here, they are 3 types of states are used, the offloading is given to the service provider and action is obtained in sets. The Markov Decision Process is done on three categories as follows.

- State: Busy, Idle, Queuing
- Action: Weight
- Probability

The actual state and time-out transactions of the proposed method is illustrated in Figs. 1a and 1b respectively.

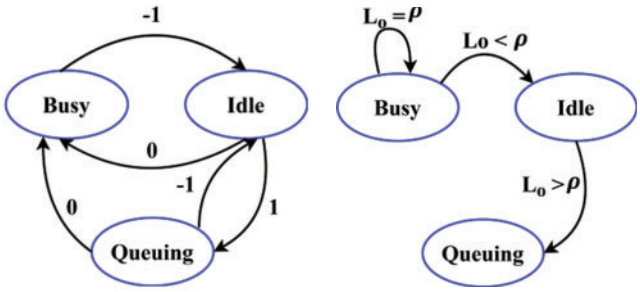

Figure 1: (a) State transaction, (b) Timeout transaction

This time-based slot is allocated for the state and action in the Markov process model. The decision is made on weights, at each step of offloading the task is one state f and the decision is chosen based on the action h, here the action is the weight. It checks whether the action is available for the state. When the task is moving to a new state means it is denoted as the probability which is represented as i. The probability for the state and action is denoted as $i(f, f_0)$. j reward is calculated by taking the current state transfer to the new state. The reward to the current and next state is referred as $j(f, f_0)$ is denoted as a new state for the task. From Eq. (4) the data transmission is considered for probability for the new state of task processing using Eq. (5)

$$d_t = \frac{u_t}{p_t} * (f_t + h_t) \tag{5}$$

From the above Eq. (5) the state and action are assigned along with data transmission model. The proposed process is derived as $i: f * h \in either\ 0, 1,\ or\ -1$ that is taking the current state and action for calculating the reward function which is having the possibility of 0, 1 or -1. Here, three sets of states are taken into consideration. Busy means there is no time slot is available to perform any task. Idle refers to no slot is available and queuing denotes there are some time slots avaliable for the task to process in the network. In this work, the offloading is done for a service provider to transfer the task. The three conditions are equated by using the following Eq. (6).

$$\gamma_{f,h} = \delta * \left[\prod_{f=0}^{h=0} L + t \right] \begin{cases} f_t * h_t = 0, & Busy \\ f_t * h_t = 1, & Queuing \\ f_t * h_t = -1, & Ideal \end{cases} \tag{6}$$

From Eq. (5) the delay and response are calculated under the maximum time slot. By using Eq. (6), the time slot is given for the following state. γ is the optimal value for deriving the task, δ is denoted as state value function. In this, the first condition is $f_t * h_t = 0$, here the state and action are added with the time slot using Eq. (6) and it is denoted as 0 means it represents the state is busy. It indicates no task is processed for offloading. In the second condition, $f_t * h_t = 1$ the queuing is performed.

Both the conditions perform the similar functions of task offloading that is to be carried out for the consecutive process. The third condition formulated as $f_t * h_t = -1$, here the negative value denotes every slot is free for processing the task. This has the particular time out process; the task is ended when it extends the time out process. By using the time out process the task is

removed from the queuing state. The following Eq. (7) is used for calculating the time out process for each state.

$$L_0 = \begin{cases} L_t + t\,(f * h) = \rho \\ L_t + t\,(f * h) < \rho \\ L_t + t\,(f * h) > \rho \end{cases} \tag{7}$$

By using Eq. (6), the three-states are obtained and from that condition, the time out is calculated by using Eq. (7). In this above Eq. (7), L_o is denoted as a time out process, L_t is time taken for the task to compute and ρ is the fixed time out for the task. The above equation is carried out in three stages using a time out. The first one $L_t + t\,(f * h) = \rho$ indicates the process is a busy state, the second condition is $L_t + t\,(f * h) > \rho$, indicates the process is in an idle state and $L_t + t\,(f * h) < \rho$, depicts the state is in queue.

According to the state, the action is assigned by concerning the weight. The task which is having high weight is denoted as a busy state if the weight is lesser means then it is in idle state. By using Eq. (7), the time out indicates to decreases the delay in the network. If the weight is equal means then the tasks are in queue. Eq. (8) is used for calculating the weight for each state. Based on the weight the state and action are used as the decision making process.

$$w\,(L_0) = \delta * \left[\prod_{f=0}^{h=0} L_t \right] \begin{cases} f + h < w \\ f + h > w \\ f + h = w \end{cases} \tag{8}$$

By using Eq. (7), the time out process is calculated for the state after the time out is calculated the weight is considered for time out. In Eq. (8) according to the weight the task is allocated by considering Eq. (5). The first case$f + h < w$ satisfies the service provider is busy. $f + h < w$ Denotes the state is in idle state and the third case $f + h = w$ is used for task processing which are in queue. By formulating Eq. (8) the service response rate increases. Fig. 2, illustrates the decision-making using Markov Model.

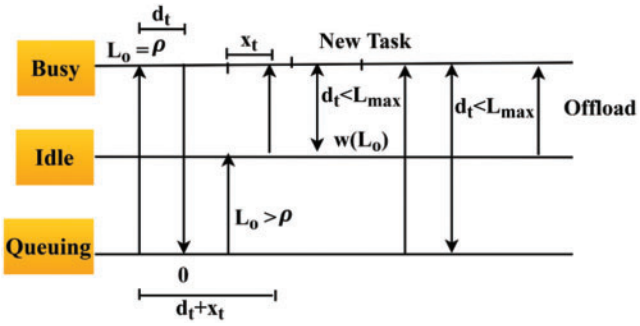

Figure 2: Decision-making in offloading

Eqs. (7) and (8) is used to satisfy the objective of improving service response by identifying time-out and weight. For a high weight-based request, the time-out is less and hence the available state resource is allocated.

4 Experiments

4.1 Experimental Setup

This section discusses the performance analysis of the proposed STOM through a comparative analysis with the existing OnLS [18] and COM [21] methods. In this analysis, the metric response rate, response time, backlogs and offloading factor are used. The comparative analysis is performed by varying the IoT requests and processing servers. The experimental setup is modeled using the ContikiKooja simulator and in Tab. 2, the experimental set and values are presented. In this experimental setup, 150 IoT devices are used for communicating with distributed cloud storage of 2TB. The request rate varies from 100–700 including 50 processing servers. The rate of request processing varies from 50/ server to 110/ server. The maximum time-out for the requests is 4.8 s from the time of acceptance by the processing server. In Figs. 3a and 3b, the flowchart for request handling and service response is illustrated.

Table 2: Experimental setup and values

Experimental setup	Value
IoT devices	150
Requests range	100–700
Processing servers	50
Request processing capacity	10 requests/instance
Maximum	2.4 s
Bandwidth	1 Mbps

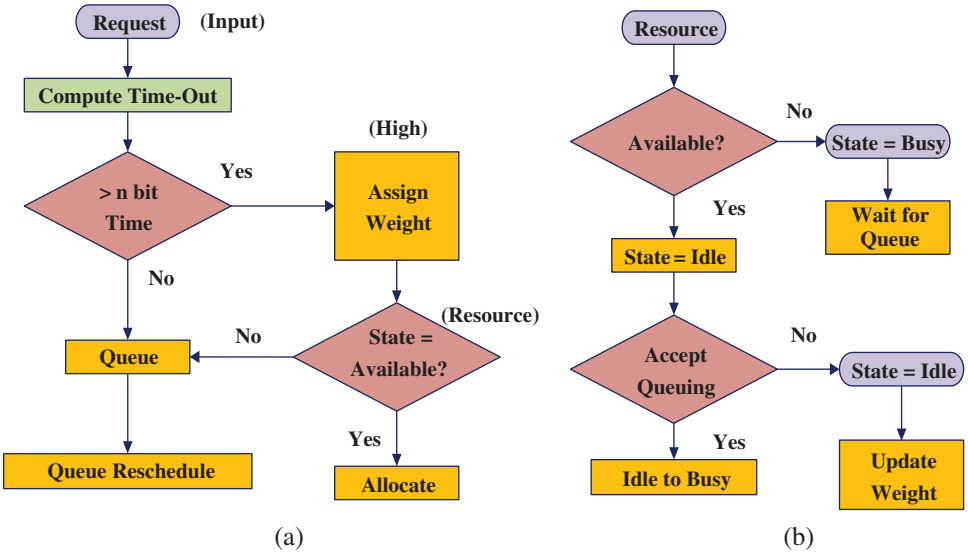

Figure 3: (a) Request processing, (b) resource allocation

The initial step of task offloading is done by calculating the processing time and data transmission by using Eqs. (1) and (2). After this, the graph is structured by using Eqs. (1) and (3). Then, the task offloading is done using Markov decision Process, and Eq. (5) is used to find the state and action probability. Eq. (6) is used for finding the optimal function. Finally, the task offloading for the service provider is obtained using Eqs. (7) and (8).

4.2 Experimental Results

4.2.1 Response Rate

The Tab. 2 illustrates the Experimental Setup and values. Response rate is the factor between the number of responses delivered and the number of requests received in a particular time intervals (say in a time "t"). Mathematically, it is defined as

$$ResponseRate = d_t/t \tag{9}$$

Here d_t is the transmitted response and t is the total number of task request received. Figs. 4a and 4b illustrates the comparative analysis of the response rate for varying requests and processing servers. X axis represents the number of servers and requests. Depending on the states of the server, the requests are allocated in appropriate time slots for processing. Conditional validation and assignment of requests to the processing server helps to prevent failures in response. The factor $\gamma_{f,h}$ is responsible for determining the states of the processing server in fore-hand using the given time slots. Therefore, $x_t(n, r)$ is the interval for slotted and allocated requests from which high response rate is observed. For the varying request, the proposed method achieves a maximum of 0.92 and a minimum of 0.893 for the requests 700 and 500, respectively. Similarly, the maximum response rate is 0.92 and the minimum is 0.904 for the processing servers 40 and 30, respectively. The response rate in the proposed method is improved by serving maximum requests irrespective of the density of the available resources. The number of variations in request to response is balanced using the available servers and offloading procedure. The state of the service provider helps to decide the allocation of resources for the available requests. Therefore, the proposed offloading method achieves a fair response rate under controlled drop/ backlog.

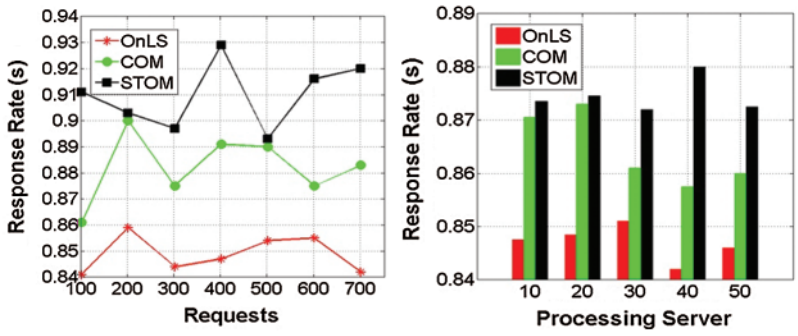

Figure 4: (a) Response rate *vs*. requests, (b) Response rates servers

4.2.2 Response Time

The response time is achieved in a less manner in the proposed method. The time required is $x_t(n, r)$ that is reduced through state assignments. The conditional verification of $d_t(d_t) < L_{max}$ retains the less time-consuming requests to be delivered within the optimal delay. The $d_t(x_t) > L_{max}$ requests are offloaded based on the states of the decision process. In this process, the assignment follows $w(L_o)$ for assigning requests to the processing server. For the varying requests, the proposed method achieves a maximum of 1.01s and 1.21s for the requests = 700 and processing server= 10. Similarly, the minimum time is 0.48 for 100 requests and 0.482 for the 50 processing servers. Therefore, the concurrent allocation of requests helps to achieve fewer response times for the varying requests and processing servers respectively [Refer to Figs. 5a and 5b].

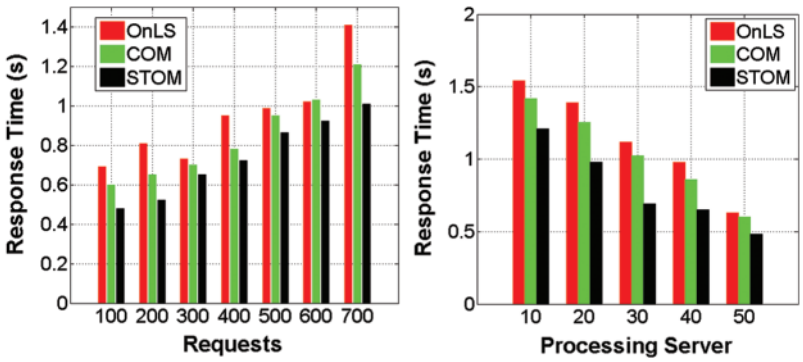

Figure 5: (a) Response time *vs*. requests, (b) response time *vs*. servers

4.2.3 Backlogs

In Figs. 6a and 6b, the backlogs in responses are compared with the existing methods. The proposed STOM achieves fair allocation of resources for the incoming requests. The state of the accepting server and the responding devices are the considerable factors for accepting or offloading the incoming requests. Therefore, the unnecessary overloading of the resources or prolonged wait time of the requests is presented in the proposed offloading method. The necessary requests are queued up in the process of resource allocation, before their actual time-out. The proposed STOM achieves a high backlog of 55 for the 700 requests and 10 processing server. On the other hand, the backlog is minimum i.e., 133 for 100 requests and 15 for 50 processing server, as seen the above figures. Therefore, the responses are high in the proposed method. Similarly, the dissemination of responses depends on the number of service to the available that reduces the backlogs. Both, the available processing server and the requests are handled in a less offloading manner to prevent additional backlogs in the proposed method.

4.2.4 Offloading Factor

The offloading factor is computed as the ratio between the differences in transmission and retained to the total number of responses. Mathematically, it is expressed as

$$\text{Offloading Factor} = (t - d_t)/(a + b) \tag{10}$$

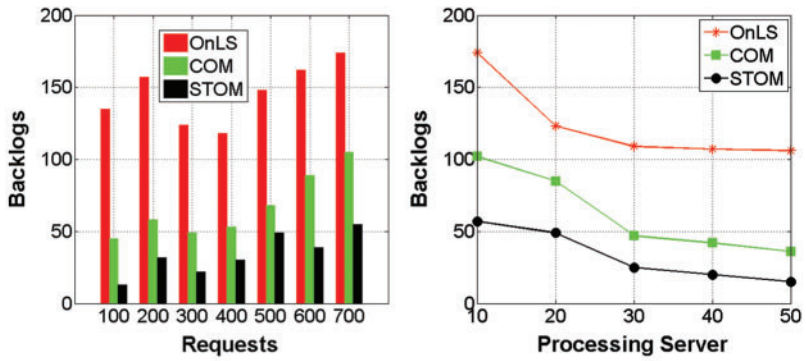

Figure 6: (a) Backlogs *vs*. requests, (b) backlogs *vs*. processing server

Here, a and b are the total responses of the task t. The comparative analysis of the offloading factor for the varying requests and processing server is presented in Figs. 7a and 7b, respectively. The proposed method does not offload all the incoming requests; rather it classifies the number of time-exceeding requests first. The classified requests are assigned with available service providers where the request to response ratio is high. This prevents conventional offloading, whereas the overloading requests that have less time-out are identified and offloaded to the service provider with the definitive state. If the state is available, the response is generated promptly else a consecutive offloading is performed, without losing the request. The proposed method is found to achieve a maximum of 0.035 offloading factor for 700 requests and 50 processing server. Whereas the offloading factor is less as 0.0079 for 100 requests and 0.0033 for 10 processing servers, respectively.

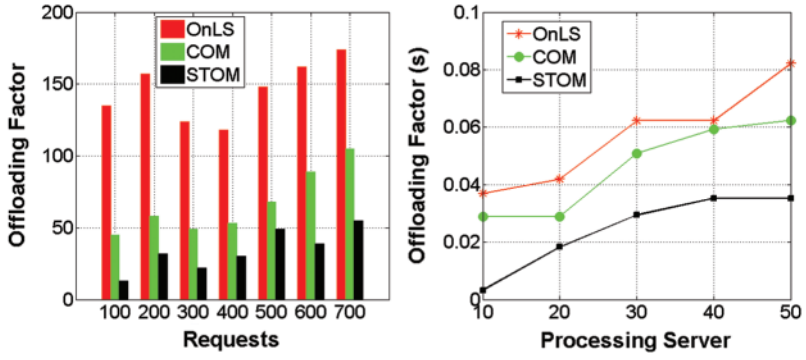

Figure 7: (a) Offloading factor *vs*. requests, (b) offloading factor *vs*. processing server

In the overloading request handling, offloading is performed in a limited manner. To prevent additional time delay, concurrent processing is performed. This helps to reduce the offloading rate in the proposed method. The comparative analysis results are tabulated in Tab. 3.

Table 3: Comparative analysis tabulation

Metrics	OnLS	COM	STOM
For Varying Requests			
Response Rate	0.842	0.883	0.92
Response Time (s)	1.41	1.2	1.04
Backlogs	174	105	55
Offloading Factor	0.0776	0.05303	0.03582
For Varying Server			
Response Rate	0.852	0.88	0.905
Response Time (s)	0.63	0.6	0.482
Backlogs	106	36	15
Offloading Factor	0.08214	0.06234	0.03529

5 Conclusion

This paper proposes a state-based task offloading method for improving the response rate of IoT services. The method defines the states based on the processing ability of the service providers and then allocates the requests. Request processing is based on the transition between the states as determined using Markov decision. The rate of processing and the time factor are the considerable factors for improving the response for the requests fetched from the IoT devices. The process of decision-making is augmented using weighted transitions from which reliable outcome of high response rate and less time is achieved. The weights are assigned for the fewer time-out requests that are handled by allocating them prior in the queue for resource allocation. This process is unanimous for the varying requests and processing server. Therefore, the response rate is retained at a high rate of irrespective of the density of the requests. The offloading decisions are made using the Markov model for admitting the requests without backlogs. The proposed method is found to achieve 11.5% high response rate, 20.31% less response time, 20.19% fewer backlogs, and 8.85% less an offloading factor. Similarly, for the varying server, the proposed STOM maximizes 11.7% high response rate and reduces response time, backlogs, and offloading factor by 21.63%, 19.72%, and 11.09%, respectively.

Acknowledgement: Authors acknowledged the following institutions for their valuable support to execute this paper. 1. Hindustan Institute of Technology and Science, Chennai, India, 2. SRM Institute of Science and Technology, SRM Nagar, Kattankulathur, Tamilnadu, India, Faculty of Engineering Built Environment, Durban University of Technology (DUT) University, South Africa.

Funding Statement: No external funding received for this research work. The partial APC is will be paid Durban University of Technology (DUT) University, South Africa.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] D. S. Park, "Future computing with IoT and cloud computing," *Journal of Supercomputing*, vol. 74, no. 12, pp. 6401–6407, 2018.

[2] Z. Hong, W. Chen, H. Huang, S. Guo and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT-edge–cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 12, pp. 2759–2774, 2019.

[3] A. Yousefpour, G. Ishigaki, R. Gour and J. P. Jue, "On reducing IoT service delay via fog offloading," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 998–1010, 2018.

[4] Y. Deng, Z. Chen, X. Yao, S. Hassan and A. M. A. Ibrahim, "Parallel offloading in green and sustainable mobile edge computing for delay-constrained IoT system," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 12, pp. 12202–12214, 2019.

[5] A. Jaddoa, G. Sakellari, E. Panaousis, G. Loukas and P. G. Sarigiannidis, "Dynamic decision support for resource offloading in heterogeneous internet of things environments," *Simulation Modelling Practice and Theory*, vol. 101, pp. 102019, 2020.

[6] H.-W. Kim, J. H. Park and Y.-S Jeong, "Adaptive job allocation scheduler based on usage pattern for computing offloading of IoT," *Future Generation Computer Systems*, vol. 98, pp. 18–24, 2019.

[7] D. G. Roy, B. Mahato, A. Ghosh and D. De, "Service aware resource management into cloudlets for data offloading towards IoT," *Microsystem Technologies*, 2019.

[8] H. Wei, H. Luo, Y. Sun and M. S. Obaidat, "Cache-aware computation offloading in IoT systems," *IEEE Systems Journal*, vol. 14, no. 1, pp. 61–72, 2020.

[9] M. Thangapandiyan, P. M. Rubesh Anand and K. Sakthidasan Sankaran, "Quantum key distribution and cryptography mechanisms for cloud data security," in *2018 Int. Conf. on Communication and Signal Processing*, Chennai, pp. 1031–1035, 2018.

[10] J. Xie, Y. Jia, Z. Chen, Z. Nan and L. Liang, "D2D computation offloading optimization for precedence-constrained tasks in information-centric IoT," *IEEE Access*, vol. 7, pp. 94888–94898, 2019.

[11] Z. Liu, Y. Yang, K. Wang, Z. Shao and J. Zhang, "POST: Parallel offloading of splittable tasks in heterogeneous fog networks," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3170–3183, 2020.

[12] L. Lei, H. Xu, X. Xiong, K. Zheng and W. Xiang, "Joint computation offloading and multiuser scheduling using approximate dynamic programming in NB-IoT edge computing system," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5345–5362, 2019.

[13] K. Sakthidasan Sankaran, N. Vasudevan and A. Verghese, "ACIAR: Application-centric information-aware routing technique for IoT platform assisted by wireless sensor networks," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 11, pp. 4815–4825, 2019.

[14] M. Aazam, S. Zeadally and K. A. Harras, "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Generation Computer Systems*, vol. 87, pp. 278–289, 2018.

[15] M. S. Hossain, C. I. Nwakanma, J. M. Lee and D. S. Kim, "Edge computational task offloading scheme using reinforcement learning for IIoT scenario," *ICT Express*, vol. 6, no. 4, pp. 291–299, 2020.

[16] K. Xiao, Z. Gao, W. Shi, X. Qiu, Y. Yang *et al.,* "EdgeABC: An architecture for task offloading and resource allocation in the internet of things, Future Generation Computer Systems, vol. 107, pp. 498–508, 2020.

[17] S. Ahmad and D. Kim, "A multi-device multi-tasks management and orchestration architecture for the designof enterprise IoT applications," *Future Generation Computer Systems*, vol. 106, pp. 482–500, 2020.

[18] S. Li, D. Zhai, P. Du and T. Han, "Energy-efficient task offloading, load balancing, and resource allocation in mobile edge computing enabled IoT networks," *Science China Information Sciences*, vol. 62, no. 2, 2018.

[19] H. Sun, H. Yu, G. Fan and L. Chen, "Energy and time efficient task offloading and resource allocation on the generic IoT-fog-cloud architecture," *Peer-to-Peer Networking and Applications*, vol. 13, no. 2, pp. 548–563, 2019.

[20] H. Shah-Mansouri and V. W. S. Wong, "Hierarchical fog-cloud computing for IoT systems: A computation offloading game," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 3246–3257, 2018.

[21] Z. Ning, P. Dong, X. Kong and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4804–4814, 2019.

[22] X. Xu, Q. Liu, Y. Luo, K. Peng, X. Zhang *et al.,* "A computation offloading method over big data for IoT-enabled cloud-edge computing," *Future Generation Computer Systems*, vol. 95, pp. 522–533, 2019.

[23] M. G. R. Alam, M. M. Hassan, M. Z. Uddin, A. Almogren and G. Fortino, "Autonomic computation offloading in mobile edge for IoT applications," *Future Generation Computer Systems*, vol. 90, pp. 149–157, 2019.

[24] A. Hussain, S. V. Manikanthan, T. Padmapriya and M. Nagalingam, "Genetic algorithm-based adaptive offloading for improving IoT device communication efficiency," *Wireless Networks*, vol. 26, no. 4, pp. 2329–2338, 2019.

[25] Y. Kotb, I. A. Ridhawi, M. Aloqaily, T. Baker, Y. Jararweh *et al.,* "Cloud-based multi-agent cooperation for IoT devices using workflow-nets," *Journal of Grid Computing*, vol. 17, no. 4, pp. 625–650, 2019.