QoS-Aware Service Provisioning in Fog Computing

Faizan Murtaza¹, Adnan Akhunzada¹,², Saif ul Islam³,
Jalil Boudjadar⁴, Rajkumar Buyya, Fellow, IEEE⁵

¹COMSATS University, Islamabad, Pakistan
²RISE SICS Vasteras AB, Sweden
³Department of Computer Science, KICSIT, Institute of Space Technology, Islamabad, Pakistan
⁴ECE, Department of Engineering, Aarhus University, Denmark
⁵Cloud Computing and Distributed Systems (CLOUDS) Laboratory School of Computing and Information System, The University of Melbourne, Parkville, Australia

Abstract

Fog computing has emerged as a complementary solution to address the issues faced in cloud computing. While fog computing allows us to better handle time/delay-sensitive Internet of Everything (IoE) applications (e.g. smart grids and adversarial environment), there are a number of operational challenges. For example, the resource-constrained nature of fog-nodes and heterogeneity of IoE jobs complicate efforts to schedule tasks efficiently. Thus, to better streamline time/delay-sensitive varied IoE requests, the authors contributes by introducing a smart layer between IoE devices and fog nodes to incorporate an intelligent and adaptive learning based task scheduling technique. Specifically, our approach analyzes the various service type of IoE requests and presents an optimal strategy to allocate the most suitable available fog resource accordingly. We rigorously evaluate the performance of the proposed approach using simulation, as well as its correctness using formal verification. The evaluation findings are promising, both in terms of energy consumption and Quality of Service (QoS).

Keywords: Fog Computing, Cloud Computing, Quality of Service, LRFC, Quality of Experience, Internet of Everything

1. Introduction

In the emerging Internet of Everything (IoE) paradigm, billions of devices are being connected to the Internet. The number, types and nature of
Internet-connected devices will also increase for the foreseeable future. For example, it was reported that more than 50 billion devices will be linked to the Internet by 2020 [1], with an estimated market worth of $7.1 trillion [2]. Quality of Experience (QoE) is one of several key metrics for the IoE users [3]. To deal with limitations inherent in a cloud computing environment (e.g. privacy of users sourcing the data to the cloud, particularly hosted in an overseas jurisdiction, and performance issues such as latency), there has been attempts to ‘push’ the computing to the edge of the network via fog nodes [4, 5]. Hence, the fog computing paradigm has been emerged to handle time/delay-sensitive IoE applications [6, 7, 8, 9, 10, 11].

As shown in Figure 1, only low-latency computational tasks are processed by the fog nodes but the computationally demanding tasks will still be routed to the cloud servers for processing [1]. Fog nodes can be any computing devices that has some computational capabilities in terms of data management, analytics, computation, storage and networking, and examples range from an intelligent home assistant (e.g. Amazon Echo) in a smart home to a military vehicle in an Internet of Battlefield Things environment. The connections between fog nodes and the cloud can be wired and/or wireless [12].

While there are many challenges associated with a fog computing environment [13], we will only focus on one of these challenges. Specifically, we will study the task scheduling problem for better fog resource management, particularly with the exponential increase in the number of IoT devices while achieving a satisfactory QoE.

The existing proposed techniques in fog computing lacks the adaptive and intelligent behavior. They only consider the direct communication of end devices preferably with closest fog nodes. However, this type of architecture only supports individual node queue level task scheduling. Further, it is quite complex to implement adaptive and intelligent learning-based task scheduling in this kind of architecture. Subsequently, we cannot take the real advantage of heterogeneity of fog nodes surrounded by a set of IoT devices. In order to facilitate intelligent and adaptive task scheduling policies in this dynamic and heterogeneous environment, we need a smart layer [14] between end devices and fog nodes that should have three main capabilities: 1) the ability to define whether the incoming request should be served by a cloud or a fog node, 2) the capability to schedule the incoming task to most appropriate fog node among the available fog devices, and 3) and for efficient task scheduling this layer should have the functionality that extensively implements adaptive and intelligent learning-based task scheduling. In this paper, we propose
Figure 1: An example fog-cloud environment.
an intelligent and adaptive task based learning technique for optimal task
scheduling in a fog computational paradigm. The key contributions of the
paper are summarized as follows:

- A proposition and implementation of a smart layer between IoE/IoT
devices and Fog nodes.

- We propose a QoS-Aware approach (hereafter referred to as the learn-
ing repository fog-cloud - LRFC). The proposed service has been provi-
sioned at multiple geographically distributed gateways deployed among
IoE devices and their corresponding Fog nodes proximity. Consequently,
making our proposed scheme highly scalable, and thus reliving poten-
tial performance bottlenecks. Finally, our proposed deployment model
significantly suits nearly all existing and futuristic environments (i.e.,
IoT, IoE, smart X (smart city, smart grid, smart building, smart forest
etc.)).

- For a comprehensive and unbiased comparison, we implement the state-
of-the-art task scheduling policies on our proposed smart layer.

- We thoroughly evaluate the performance of the proposed approach us-
ing simulation, and prove its correctness through formal verification.
The proposed approach shows promising results in terms of processing
delay, overall network propagation time and power consumption.

The remainder of this paper is organized as follows. In the next section,
we present the background of the proposed approach and relevant works.
Section 3 describes the complete methodology (i.e., system model and algo-
rithms) of the proposed approach. The formal verification of our proposed
scheme is detailed in Section 4. Performance evaluation is comprehensively
elaborated in Section 5. Finally, Section 6 concludes the paper along with
thoughts for future work.

2. Related Work

The section briefly explains the background and related research.
2.1. Learning Based Approaches

Rule-based learning is the simplest form of artificial learning [15]. In general, rules are expressed in IF-THEN condition, and they can be simple or multi conditional IF-ELSE statement [16] [17]. It has been shown that rule-based learning can be applied in a wide range of applications, such as to reduce and predict the energy gap between predicted and actual energy consumption in buildings [18], big data classifications problems [19], user behavior classifications [20], developing decision support system for risk assessment and management strategies in distributed software development [21], and so on.

Case-based Learning (CBR), a lazy learning approach, works well where rich structured knowledge is not available earlier (e.g. in autonomic computing ) [22] [23]. For learning purpose, it involves the processing of a set of cases to train the system and predict values from these cases (and prior knowledge). In [24], for example, the authors proposed an architecture for sharing of experience using an agent-based system architecture layout (SEASALT). The latter works with diverse data repositories to maintain, retrieve, adopt and retain cases. Similarly, in [25], the authors proposed a temporal CBR for diabetes insulin by looking at prior events such as blood pressure level, physical activities and carbohydrate ingestion. Additionally, similarity metrics like Euclidean distance have also used with CBR for pattern classification [26].

Hybrid approaches can be more effective than either a rule- or case-based learning approach to solve complex problems [27],[28]. In [29], the author used a hybrid approach for domain-independent clinical decision support in a hospital’s intensive care unit. The potential for a hybrid approach was demonstrated in a medical diagnosis system by the authors in [30] and [31].

2.2. Task Scheduling in Fog-Cloud Environment

Efficient task scheduling in fog computing to maximize resource management is one of several research focuses in recent years [32, 33]. We mainly classify the task scheduling approaches into two main categories in Fog-Cloud environments.

2.2.1. QoS-aware Task Scheduling in Fog-Cloud Environment

In [7], the authors proposed a bio-inspired optimization approach (i.e., Bees Life Algorithm (BLA)), seeking to address the job scheduling challenge in a fog computing environment. The approach is based on the optimized
distribution of a set of tasks among fog nodes to deal with users' excessive requests to computational resources. This approach also seeks to minimize energy consumption and CPU execution time. In a different work, the authors of [9] proposed three different strategies for optimum resource utilization. Firstly, a random methodology is used to select the fog nodes to execute tasks upon arrival. Secondly, the focus will be on the lowest latency fog devices. Finally, the fog resources having the maximum available capacity should be primarily considered. The three proposed policies were then evaluated using a mathematical model. The authors in [10] introduced a joint optimization task scheduling and image placement algorithm, which is designed to minimize the overall completion task time for better user experience. The first sub-problem investigated is to balance the workload both on client devices and computational servers. The second sub-problem investigated is on the placement of task image on storage servers, and the final sub-problem is to balance the input-output interrupt requests among the storage servers. The research [34] discusses applications scheduling in the fog computing process and focuses on the influence of user mobility on application performance. Different policies have been used in scheduling, when different application requests arrive these policies decides to execute them on cloudlet or on the cloud. Policies include concurrent policy, in this policy all request received at the cloudlet or allocated to cloudlet without measuring the usage. First Come First Server (FCFS) policy works in a traditional way serving the requests on arrival until it consumes all resources. The third was Delay-priority policy in which all request that requires lower delay was scheduled first. In [6], the authors proposed a workload allocation framework to balance the computational latency and power consumption in a fog-cloud environment. Likewise [6], the authors in [10] studied the trade-off between power consumption and computational latency in fog. The approach is based on the convex optimization technique such as interior-point method [35], the mixed integer nonlinear programming problem using generalized benders decomposition [36], and the Hungarian method [37] to address the problem in a fog-cloud environment. Furthermore, A. Toor et al. [38] proposed and evaluated energy and performance aware fog computing scheme using Dynamic Voltage Frequency Scaling (DVFS) technique while utilizing the green renewable energy resources. Similarly, effective resource utilization and computing service of delay sensitive applications were considered in [39]. The study proposed a graph representation base system and a task-oriented dynamic load balancing algorithm that maps the physical resources to virtualized re-
sources. Each resource is represented by a node and has a certain capacity.
On the arrival of a new fog node, the algorithm reallocates the load in its
nearby neighborhood to maintain the balance and accounting task distribu-
tion degree and the links among nodes. A reverse strategy was adopted to
remove edges not having sufficient resources.

2.2.2. QoE-aware Task Scheduling in Fog-Cloud Environment

QoE refers to the user experience towards the various service aspects.
It considers user needs, perceptions and intentions regarding provided ser-
dices [3]. Our published work [3] mainly focuses on QoE-aware task schedul-
ing in Fog computing environment. The article uses a learning fuzzy logic
based approach to enhance QoE in hierarchical, distributed and heteroge-
neous Fog-IoT environment. The technique follows the prioritized applica-
tion placement to the suitable Fog servers using fuzzy logic models. Similar
to [6], the authors in [40] also studied load balancing while focusing on qual-
ity of experience (QoE). The proposed algorithm uses clustering in order to
meet the computation demands and minimize the power consumption. The
first in first out (FIFO) mechanism is used for task scheduling and earliest
deadline first (EDF) policy is used for resource allocation. The authors in [41]
and [42] considered multiple factors and formulated resource management on
the basis of changing the relinquishing probability of the customer, service
type, service price. However in [42], resources were taken into account on the
nature of devices. A loyalty based task scheduling model, a service-oriented
resource management model to perform efficient and fair management of re-
sources for IoT deployment, was proposed that incorporates the users history
of resource usage to increase the fairness and efficiency, when the resources
were actually consumed.

Table 1 summarizes the literature discussed in this section.
<table>
<thead>
<tr>
<th>Ref</th>
<th>Year</th>
<th>Problem</th>
<th>Methodology</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3]</td>
<td>2019</td>
<td>QoE in the hierarchical, distributed and heterogeneous Fog-IoT environment</td>
<td>Prioritized application placement to the suitable Fog servers using fuzzy logic models</td>
<td>QoE-aware application placement</td>
<td>Energy consumption is not evaluated</td>
</tr>
<tr>
<td>[34]</td>
<td>2017</td>
<td>Scheduling in the fog computing.</td>
<td>Concurrent, FCFS and Delay-Priority scheduling policies.</td>
<td>CPU execution time and delay</td>
<td>Non cooperative simulation environment</td>
</tr>
<tr>
<td>[39]</td>
<td>2016</td>
<td>Resource utilization and computing service of delay sensitive applications</td>
<td>Graph representation base, task-oriented dynamic load balancing algorithm that maps the physical resources to virtualized resources</td>
<td>Network flexibility, Dynamic load balancing</td>
<td>High load balancing complexity</td>
</tr>
<tr>
<td>Ref</td>
<td>Year</td>
<td>Problem</td>
<td>Methodology</td>
<td>Pros</td>
<td>Cons</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>[40]</td>
<td>2015</td>
<td>QoE in terms of fog load balancing.</td>
<td>Low complex task scheduling algorithm</td>
<td>Energy efficiency, Task latency, Power consumption</td>
<td>High complexity for large scale fog architecture.</td>
</tr>
<tr>
<td>[41]</td>
<td>2015</td>
<td>Resource management on the basis of changing relinquish probability of the customer, service type, service price, and variance of the relinquish probability</td>
<td>Loyalty base task scheduling algorithm</td>
<td>Resource management</td>
<td>Scalability</td>
</tr>
<tr>
<td>[42]</td>
<td>2015</td>
<td>Resource allocation</td>
<td>Loyalty base task scheduling model, a service-oriented resource management</td>
<td>Resource management, Performance measure</td>
<td>The large-scale request is ignored.</td>
</tr>
<tr>
<td>[43]</td>
<td>2015</td>
<td>Exploiting local resources in data stream processing</td>
<td>An extension to storm open source data stream processing as QoS-aware scheduler.</td>
<td>Network latency, Inter-node traffic</td>
<td>Availability of applications, Not suitable for composite fog computing</td>
</tr>
</tbody>
</table>
In today's dynamic and heterogeneous environment, it necessitates a smart layer between end devices and fog nodes to encourage intelligent and adaptive task scheduling approaches. The main capabilities of the smart layer should include: a) the capacity to define whether the approaching request ought to be served by a cloud or a fog node, b) the layer should be able to assign the incoming task to the most fitting fog node among the accessible fog nodes in a geographical proximity; and c) to extensively implement adaptiveness and learning-based intelligence for varied task scheduling processes.

3. Methodology

The methodology of the proposed scheme is detailed below.

3.1. System Model

In this paper, we have designed the complete architecture of the Fog Cloud system as shown in 2. The architecture contains three layers. The first layer comprises of several IoE devices generating N number of requests. The second layer includes geographically distributed gateways deployed among the proximity of IoE devices and their corresponding Fog nodes. Provisioning of our proposed service at varied geographically distributed gateways makes our technique highly scalable and thus avoiding performance bottleneck. The proposed deployment model suits the existing and futuristic environments (i.e., IoT, IoE, smart X (smart city, smart building, smart forest etc.)).

3.2. Learning Repository Fog-Cloud

To ensure effective job scheduling, we propose an intelligent and adaptive approach, named, Learning Repository Fog-Cloud (LRFC) that is a solution deployed at various gateways in the second layer. The basic sequence and operations of our proposed system are shown in Figure 3. The sequence starts with the generation of asynchronous tuples to the LRFC layer. Jobs are decomposed into tasks in second step. Further, the Learning Repository creates its Meta in step three. In step four, best fitted fog servers are selected to serve the jobs. Hence, the LRFC schedules the task either for Fog or Cloud (i.e., if no suitable fog server is found or all the fog servers are completely occupied, then the tasks will be sent to cloud for further processing) in step five. In step six, Fog executes the task and returns the response details to LRFC layer; whereas, the cloud executes the received requests and returns response to LRFC layer in step seven, respectively. LRFC receives
Figure 2: High level architecture.
the response and updates the information iteratively in step eight. Finally, the results are generated accordingly.

3.3. An Adaptive and Intelligent Task Scheduling Approach

We propose a hybrid approach based on the idea of both rule-based learning and case-based reasoning to produce efficient results. For this purpose, a learning repository is created that stores the particulars of each incoming task such as tuple identification (ID), tuple type, and the information of the resource where the task is served. Moreover, it also maintains the information such as propagation time, execution time, energy consumption of the tuple at a specific resource. The tasks are scheduled to the available resources based on the information stored in learning repository. Additionally, the selection of the service type (i.e., Fog, Cloud) is also made through the learning
repository information. The learning repository is regularly updated, and the scheduling decisions are made accordingly. We have a \( tp_{total} \) total number of tuples and \( fg_{total} \) total number of fog servers. Initially, we are creating learning repository. Five percent of the \( tp_{total} \) are used for training our proposed approach. The remaining ninety-five percent (95\%) of \( tp_{total} \) is used for testing purpose. Algorithms 1 and 2 provide a detailed and self-explanatory description of our proposed approach. The learning repository storage \( F_S \) and \( C_S \) are created for Fog and Cloud, respectively. The initial tuples used for training are abbreviated as \( tp_{ini} \). The tuples used for are testing are presented as \( tp_{final} \). The \( SR \) stands for Storage Repository. Whereas, \( DT \) represents the type of IoT job. Internal Processing Time \( ITP \) is the time that a tuple/job takes for processing using a resource of Fog or Cloud. The link propagation time is abbreviated as \( PT_L \). In the \( LR_{exe} \) process, the selection of fog server is decided on the distance between the request generated from and the server location where it is physically placed. The nearest server is selected with its server id \( fg_{id} \). The \( fg_{id} \) of the serving server will be saved in \( LR \) storage as well. In the case of cooperation, if the selected server \( fg \) is busy then the tuple will wait in the queue or will be sent to the Cloud.

4. Formal Verification of the Proposed Scheme

In this section, we analyze the efficiency of our scheduling algorithm in a formal way. To achieve that goal, we have used Uppaal timed automata to describe the behavior or IoT devices and scheduling algorithm. Then used the underlying model checker to verify a set of derived properties. Given the rich expressiveness of Uppaal formalism, the behavior of IoT devices, fogs and algorithm can be constrained with time attributes such as the response time of the scheduler to acquire a request, minimum/maximum time interval between two requests from the same IoT device, time to communicate with a fog, etc.

Request types describe the combination of the resource types that an IoT device can request. A request type can include a single resource (e.g. computation resource), or a combination of different resources (e.g. computation and storage resources). We use \( ReqT \) to denote a request type. A request \( R \) targets a set of resources each with a different amount, e.g. \( R=(Compute=500, RAM=12) \). We use notation \(|R^i|\) to refer to the amounts of individual resources of the request \( R \). As example, \(|R^1|= (Compute, 500)\).
Algorithm 1 Learning Repository Fog-Cloud Approach

procedure LRFC($tp_{total}$, $fg_{total}$)
    $C_T = \{true, false\}$
    $S_t = \{S_1, S_2, S_3\}$
    $D_c$
    $C_o = \{true, false\}$
    $G_w = \{true, false\}$

    $F_s, C_s$ > storage for fog and cloud

    $tp_{ini} = (tp_{total} \times 5/100)$ > getting 5% to initial execute
    $tp_{final} = (tp_{total} \times 95/100)$ > 95% of remaining

    split = Splitting $tp_{ini}$ into 5 chunks

    for ( j = 0; j < split.length ) do
        for ( $t_i$ in split[j] ) do
            $Rd_{num} = [0, 1]$

            if ( $Rd_{num} = 0$ ) then
                $fg_i = exec FOGSIM( f_{g_{total}}, t_i, C_T, 0 )$
            else
                $S_i = get service from S_t$
                $SendToCloud( t_i, false, S_i, D_c )$
                $temp = \{ t_i, D_T, t_i.ID, t_i.IPT, t_i.PT, D_c.Name, t_i.Link \}$
                $C_s + = temp$
            end if
        end for
        $j +=$
    end for

    for each ( $t_i$ in $tp_{final}$ ) do
        $F_{F_s} = getting t_i from F_s$ on the basis of $D_T$ and order by $t_i.IPT \land t_i.PT_L$
        $C_{C_s} = getting t_i from C_s$ on the basis of $D_T$ and order by $t_i.IPT \land t_i.PT_L$
        $Boolean f, c$
        $f = set true if F_{F_s} is null$
        $c = set true if C_{C_s} is null$
        if ( $F_{F_s}$ is null OR $C_{C_s}$ is null ) then
            if ( f ) then
                exec $FOGSIM( f_{g_{total}}, t_i, C_T, 0 )$
            end if
        end if
    end for
Algorithm 2 Learning Repository Fog-Cloud Approach - Part 2

if (c) then
    SendToCloud( t_i, false, S_i, D_c)
    temp = {t_i, D_T, t_i.ID, t_i.ITP, t_i.PT, D_c.Name, t_i.Link}
    C_s+ = temp
else
    C_{cost} = C_{Cs}.IPT + C_{Cs}.PTL
    F_{cost} = F_{Fs}.IPT + F_{Fs}.PTL
    if ( C_{cost} >= F_{cost} ) then
        exec FOGSIM( f_g_{total}, t_i, C_T, F_{cost}.f_g_{Id})
    else
        SendToCloud( t_i, false, S_i, D_c)
        temp = {t_i, D_T, t_i.ID, t_i.ITP, t_i.PT, D_c.Name, t_i.Link}
        C_s+ = temp
    end if
end if

end procedure

function FOGSIM( f_g_{total}, t_p_i, C_T, F_{cost}.f_g_{Id})
for each ( f_g in t_p_{final} ) do
    if ( f_g hasPower() ) & & hasResources() ) then ▷ for requested tuple ti with f_g_{Id}
        CalcDisMinProximity()
        f_g_i = f_g
        if ( f_g_i = null & & C_o = true ) then
            CalcDisMinProximity()
            f_g_i = f_g
        end if
    end if
end for
    temp = {t_i, D_T, t_i.ID, t_i.ITP, t_i.PT, f_g_i.ID, t_i.Link}
    F_c  += temp
end function
Figure 4 shows a parameterizable model of IoT devices. A device behavior is initially at location Init, waiting for certain time interval before issuing a request. The request type and budget are randomly generated using function Fresh(). While performing a request, the IoT device synchronizes with the scheduler using event Demand!. The IoT device waits then to be scheduled to a fog at location Ready. Whenever scheduled, the device waits until the request is fully satisfied upon event Terminate[]? then it can start another request. Function Expired(Req) calculates when a request gets satisfied, given the performance characteristics of the fog and the budget requested by the IoT device.

![Diagram of IoT device]

Figure 4: Template of the IoT device.

To simulate the learning process, we use dynamic priorities for the allocation of fogs to different request types. The priority to allocate a fog $F$ to a request type changes on the fly according to the dynamics of the system. Initially all priorities are set to zero ($Prio(ReqT,F,0)=0$), meaning that there is no preference in allocating a given fog to a given request type. For a given request type $ReqT$, if a fog $F$ has recently been used many times to satisfy requests of type $ReqT$ then the priority to allocate $F$ to the given request type gets increased over time. Otherwise, the priority will be decreased on each time interval of length $\delta$ if $F$ has not been allocated to any request of the concerned type during the last $\delta$ time interval.

$$Prio(ReqT,F,t) = \begin{cases} 
Prio(ReqT,F,t-\delta) + + & \text{if } \exists (R_1, \ldots, R_n) \in \text{ReqT}, \\
& \exists t \in [t-\delta,t], \\
& \wedge \text{Allocate}(R_i,F,t')=\text{True} \\
\text{Prio}(ReqT,F,t - \delta) - & \text{Otherwise} 
\end{cases}$$

Figure 5 shows the learning model where for each time interval delta,
function \textit{All\_Prio(delta)} calls the aforementioned function \textit{Prio()} for all potential fogs and request types as parameters.

![Diagram](image)

\textbf{Figure 5: Template of the learning process.}

When receiving a request from an IoT device, our scheduler considers first fogs having high priorities for the allocation to that request type. If two fogs have the same priority to serve a request and both have sufficient available resource budget, then the scheduler considers the neighborhood attribute where the closer fog gains the allocation. The allocation considers the resource availability at the fog level, the learning-based priority and the neighborhood. The neighborhood attribute is calculated based on the geolocation of the device and fog.

The following function is used to show how to check whether a fog \( F \) satisfies a given request \( R \) issued at time instant \( t \).

\[
\text{Satisfy}(R, F, t) = \begin{cases} 
\text{True} & \text{if } \forall R^i \in R \quad \text{budget}(|R^i|, t) \text{budget}(F, |R^i|, t) \\
\text{False} & \text{Otherwise}
\end{cases}
\]

Function \textit{budget}(X, t) returns the amount requested by \( R \) for a resource type \( X \) at time instant \( t \). We overload this function to return the available amount of a given resource type \(|R^i|\) in a fog at time instant \( t \), simply writing \( \text{budget}(F, |R^i|, t) \).

The allocation of a fog to satisfy a request is performed via the following function:

\[
\text{Allocate}(R, F, t) = \begin{cases} 
\text{True} & \text{if } \text{Satisfy}(R, F, t) \land \forall F_j \text{ Satisfy}(R, F_j, t) \Rightarrow \\
& \text{not}(\text{Prio}(R, F, t) \land \text{Prio}(R, F_i, t) \land \\
& \text{Near}(F_i, \text{IoT}(R)) \land \text{Near}(F, \text{IoT}(R))) \\
\text{False} & \text{Otherwise}
\end{cases}
\]

Function \textit{IoT}(R) returns the actual IoT device that performed request \( R \). So that we calculate the neighborhood \( \text{Near}(F_i, \text{IoT}(R)) \) of the request issuer to the fog.
Figure 6 shows our scheduler model. Basically, whenever a device issues a request it asks the scheduler to reserve a fog to it upon the event Demand[]. If there exists a fog satisfying function Allocate() for the given request, the fog will be immediately reserved using event Reserve[], from location Scheduling. If such a fog does not exist, at location Not_Optimal, the scheduler searches a fog that satisfies the request while being the most used one to serve such a request type recently. Otherwise, the scheduler searches a fog which is the nearest one to the IoT device. If none of the cited options exists, the scheduler mediates the request to the cloud.

![Diagram of the scheduler model](image)

Figure 6: Template of the scheduler model.

When a request processing terminates, the corresponding fog resources are released and become available. We omit describing the termination function as it is very trivial.

Figure 7 shows the fog model. It is simple and consists in receiving a Reserve[] event from the scheduler, synchronizing with the requesting IoT device to start supplying resources. The fog updates the availability of its resources according to the current request budget, both when starts and terminates.

The efficiency property we have formally analyzed using model checking is the following:

**Property 1.** Each request, from an IoT device, is always satisfied by the
nearest fog having sufficient budget and an experience to satisfy such type of requests. Formally, we write:

\[
\forall R \text{ Allocate}(R, F_j, t) \Rightarrow \neg (\exists F_k \mid Satisfy(R, F_k, t) \\
\land \text{Near}(F_k, IoT(R)) \land \text{Near}(F_j, IoT(R)) \\
\land \text{Prio}(R, F_i, t) \land \text{Prio}(R, F_k, t))
\]

5. Performance Evaluation

The simulation environment is set up using CloudSim [44] and iFogSim [45]. The arguments on why using CloudSim and iFogSim in fog/edge computing environments has been detailed in [13]. Modelling of environment is inspired by Azure Cloud Service [46], and Amazon S3 service [47]. Analysis is performed based on relevant parameters such as data generation from IoTs [48] data type, internal processing time, total processing time, queue delay, propagation delay, power consumption, available resources and resources required by a tuple, cooperation of fog nodes, and distance measurement in kilometers between nodes. The cloud services in simulation setup are presented in Table 2. Moreover, cloud data centers considered in simulation environment are shown in Table 3. Furthermore, the detail about dataset, deployment of fog and IoE devices, specifications of fog servers and performance evaluation metrics is given below:

5.1. Dataset

Figure 8 presents the categories of the dataset used in the simulation [49]. It is a synthetic dataset. We have considered 30 and 50 thousand tuples to perform the experiments. The x-axis shows the different types of tuples and
Table 2: Cloud services considered in the simulation setup

<table>
<thead>
<tr>
<th>Service</th>
<th>RAM (MB)</th>
<th>Storage (MB)</th>
<th>MIPS</th>
<th>Data Rate (kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service 1</td>
<td>8192</td>
<td>5000</td>
<td>9000</td>
<td>500</td>
</tr>
<tr>
<td>Service 2</td>
<td>16384</td>
<td>10000</td>
<td>18000</td>
<td>500</td>
</tr>
<tr>
<td>Service 3</td>
<td>32768</td>
<td>15000</td>
<td>27000</td>
<td>500</td>
</tr>
</tbody>
</table>

y-axis presents the number of tuples. This dataset is used to evaluate the performance of the proposed scheme.

5.1.1. Dataset Characteristics

Dataset consists of multiple tuples. These tuples contains various properties such as - size, bandwidth, MIPS, and memory. Where, the tuple size refers to the required size of the tuple. It is important in terms of memory consumption at fog server and processing is performed based on the tuple size. The bandwidth of a tuple defines the bandwidth required by the tuple to reach its destination fog server. Every tuple has its specific bandwidth requirements and should be handled accordingly. Tuple MIPS refers to the processing requirements of a tuple to be executed at fog server. The memory of a tuple exhibits the memory required (in megabytes) by a tuple at a fog server. The dataset contains the job of heterogeneous nature having the information - name of the tuple, Tuple ID, tuple size, MIPS (required by a tuple), bandwidth, location (coordinates), type of IoE job (for instance, textual tuple of small size, medical tuples and abrupt, etc.), IsReversed (if not served by any resource), IsServed (it has a boolean value and defines whether the tuple is served or not by any available resource - cloud/fog), IsServerbyCloud (exhibits if the tuple is served by the cloud resource), type of the device (sensor, mobile, actuator etc), queue delay (the time a task rests in queue), processing time etc. The dataset can be used in varied computing environments.

5.2. Geographic Deployment

Fog servers and IoE devices are deployed randomly at varied latitude and longitude of Rawalpindi and Islamabad cities, in the simulation setup. The specifications of the fog servers are presented in Table 4. Whereas, Cloud data centers are deployed in Singapore and United States of America (USA).
Figure 8: IoE dataset used in simulation.

<table>
<thead>
<tr>
<th>DC</th>
<th>Geo Location</th>
<th>Memory (MB)</th>
<th>Storage (MB)</th>
<th>MIPS</th>
<th>BW (kbps)</th>
<th>Arch</th>
<th>OS</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA Data-center</td>
<td>37.422421,-22.0866703</td>
<td>51200</td>
<td>1000k</td>
<td>500k</td>
<td>50k</td>
<td>x86</td>
<td>Linux</td>
<td>Live</td>
</tr>
<tr>
<td>Singapore Data-center</td>
<td>1.277911, 103.849662</td>
<td>51200</td>
<td>1000k</td>
<td>500k</td>
<td>50k</td>
<td>x86</td>
<td>Linux</td>
<td>Live</td>
</tr>
</tbody>
</table>
Table 4: Fog servers specifications

<table>
<thead>
<tr>
<th>Fog_ID</th>
<th>Ram (MB)</th>
<th>MIPS</th>
<th>UpBw (kbps)</th>
<th>DownBw (kbps)</th>
<th>No. of Cores</th>
<th>Longitude</th>
<th>Latitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-0</td>
<td>10240</td>
<td>85000</td>
<td>1500</td>
<td>1200</td>
<td>2</td>
<td>72.94457212</td>
<td>33.74430192</td>
</tr>
<tr>
<td>F-1</td>
<td>8192</td>
<td>75000</td>
<td>1200</td>
<td>1000</td>
<td>2</td>
<td>72.99649278</td>
<td>33.57278</td>
</tr>
<tr>
<td>F-2</td>
<td>8192</td>
<td>75000</td>
<td>1200</td>
<td>1000</td>
<td>2</td>
<td>73.03918583</td>
<td>33.78799432</td>
</tr>
<tr>
<td>F-3</td>
<td>10240</td>
<td>85000</td>
<td>1500</td>
<td>1200</td>
<td>2</td>
<td>72.89806118</td>
<td>33.75918676</td>
</tr>
<tr>
<td>F-4</td>
<td>16384</td>
<td>110000</td>
<td>2500</td>
<td>1700</td>
<td>4</td>
<td>73.05820695</td>
<td>33.58553272</td>
</tr>
<tr>
<td>F-5</td>
<td>6144</td>
<td>50000</td>
<td>1000</td>
<td>700</td>
<td>1</td>
<td>72.95355966</td>
<td>33.73420675</td>
</tr>
<tr>
<td>F-6</td>
<td>10240</td>
<td>85000</td>
<td>1500</td>
<td>1200</td>
<td>2</td>
<td>73.05820695</td>
<td>33.58553272</td>
</tr>
<tr>
<td>F-7</td>
<td>16384</td>
<td>110000</td>
<td>2500</td>
<td>1700</td>
<td>4</td>
<td>72.95355966</td>
<td>33.73420675</td>
</tr>
<tr>
<td>F-8</td>
<td>8192</td>
<td>75000</td>
<td>1200</td>
<td>1000</td>
<td>2</td>
<td>73.01082938</td>
<td>33.70381249</td>
</tr>
<tr>
<td>F-9</td>
<td>12288</td>
<td>95000</td>
<td>2000</td>
<td>1500</td>
<td>3</td>
<td>72.89806118</td>
<td>33.75918676</td>
</tr>
</tbody>
</table>

5.3. Evaluation Metrics

The following evaluation metrics have been considered to evaluate the performance of the proposed scheme. Processing Time: The time taken by a tuple/job for processing at a fog server is termed as processing time. We compute the processing time using Equation 1.

\[
T_p = \frac{P_i(t)}{CP_i(fs)} \tag{1}
\]

The $T_p$ refers to the total processing time. Where, $P_i(t)$ is the power of the $i^{th}$ and $CP$ stands for the current power of a fog server $fs$. Response
time: The round trip time of a tuple when it is generated from the source and returned back after completion. The following Equation 2 shows the computation of response time. Where, \( RT \) stands for response time, \( T_p \) is the propagation time, \( T_p \) shows processing time and \( T_q \) presents the queue delay.

\[
RT = T_p + T_p + T_q
\] (2)

5.4. Policies

Here, following IoE data generation policies are considered to evaluate the proposed scheme. **Random policy:** This policy sends the tuples to fog servers arbitrarily without following their **First-Come-First-Served:** It refers to the synchronous forwarding of the tuples to fog servers [34] according to their order of generation from IoE devices. **Shortest-Job-First:** SJF policy sends the small tuples on higher priority to fog servers in comparison to other tuples.

5.5. Results

**Fog Servers Utilization:** Figure 9 shows the utilization of Fog resources. Figure 9.a shows the simulation time (in seconds) at x-axis and y-axis represents the number of tuples a Fog server is serving. Each Fog server has specific computing capacity in terms of MIPS. Similarly, the incoming job also has a specific size in terms of MIPS. The size, mean-inter-arrival time and the change in number of tuples directly affect the utilization of a Fog device. Figure 9.b shows utilization of a Fog server in 100 seconds of simulation time. Figure 9.c shows the utilization of 7 Fog servers. The x-axis presents fog devices and y-axis shows their percent utilization. In case of LRFC, the minimum utilization of Fog servers is 86.295 and it reaches the 90.435 at maximum. Servers utilization is used as a key metric to determine the resource utilization in Fog environment. Hence, it has a main role in resource management. Subsequently, the utilization of Fog resources has a direct impact on the their energy consumption. The under-utilization of Fog servers results in wastage of resources. Consequently, the efficient utilization of distributed resources is crucial to reduce the energy and to enhance the performance in fog environment.

**Power and Energy Consumption:** Figures 10.a and 10.b exhibits the power and energy consumption of fog resources respectively. Two types of power consumption occurs in fog servers - static power consumption and
Figure 9: Fog servers utilization.
dynamic power consumption. When a fog server is powered-on and it has no load, it consumes a constant amount of power that is required by its hardware and software for basic functions is called static power. The rest of its power consumption is proportional to its utilization - known as dynamic power consumption. In the experiments, both types of power consumption is computed. The Figure 10.a presents the average power consumed by fog resources while applying different types of task scheduling policies considered in our simulation environment. It can be seen that the proposed policy exhibits the lowest power consumption in comparison to the rest of the policies. It is due to the fact that LRFC allocates the resources according to the requirements of the jobs that ultimately has an impact on the overall power consumption of fog resources. The efficient task scheduling results in reduced power consumption in distributed environment [50]. Similarly, the energy consumption (that is the power consumed in a specific time period) is also considerably decreased while allocating resources efficiently. Most of the state-of-the art policies concentrate on the load-balancing that improves the quality of service (QoS); however, the energy consumption is somehow compromised. Moreover, allocating resources without considering nature of the jobs results in inefficient resource utilization that ultimately increases their power and energy consumption. Similarly, the overall computation time is also increased if the inappropriate resources are allocated. The LRFC policy selects the resources that perform best for specific types of jobs. This activity decreases the computation time that consequently minimizes the energy consumed by fog servers. The FCFS policy exhibits the highest energy consumption as it simply follows the flow of traffic and lacks in selecting the best resources. The shortest job first policy performs better than FCFS as it initially serves the smaller jobs. Subsequently, the smaller jobs do not wait and are served timely. In FCFS, smaller and bigger jobs are served at the same time and resources are occupied by bigger jobs that increase the overall computation time and hence the energy consumption. The random policy creates a load-balancing in the overall system and performs better than FCFS and SJF. The LRFC shows the optimal results because of the desirable task scheduling according to the job requirements.

Figure 11.a shows the average propagation delay in cloud-only and cloud-fog environments. It is clear that Fog-cloud environment is clear winner in terms of propagation delay. It is due to the remote deployment of cloud resources that create a huge latency in flow of jobs generated far from the cloud infrastructure. Contrarily, Fog exists nearer to the IoT devices and
significant reduces the delay [51].

Whereas, Figures 11.b and 11.c show a comparison of end-to-end delay of IoE jobs served on cloud-only and fog-cloud environments. The y-axis of both Figures present the end-to-end delay in milliseconds. It exhibits the usage of network and processing resources by IoE jobs. As, the cloud is remote from the end devices so the jobs traverse all the network and and use all underlying network resources and bandwidth. Whereas, fog is closer to devices where jobs are generated and results in lower utilization of network resources. In the given scenario, majority of the requests are served by fog that not only results in lower latency but it also reduces the burden of traffic on cloud.

Figure 12.a presents the processing delay occurred at the Fog servers while serving various IoE requests and applying different task scheduling policies. The x-axis presents the different task scheduling policies; whereas, y-axis shows the processing time in milliseconds. The strategy of allocating available Fog resources to incoming jobs has an important impact on the overall processing delay. If the jobs are not placed optimally at the fog resources, it causes the resource contention that increases the processing delay. Moreover, electing the best resource according to the job requirements improves the performance of the system by decreasing the delay. Figure 12.a presents that when resources are allocated dynamically and intelligently, it reduces the server level processing delay. Hence, LRFC outperforms rest of the policies.
Figure 11: A comparison of Cloud-only and Fog-cloud environments in terms of Propagation Time and End-to-end Delay.
considered in our simulations, in terms of overall average processing delay.

Figure 12.b presents the average propagation time taken by all IoE jobs while traveling in the network. It includes the time taken in transmission of the data at links, at routers and switches and also in fog-to-fog communication in case of cooperation among fog servers. The Figure 12.b clearly depicts that LRFC policy shows the better results compared to Random and FCFS; however, a slight increase is noted compared to SJF policy. Initially, LRFC takes some time to be matured and jobs are assigned randomly to the fog resources. However, it gets smarter with the passage of time. The prematurity time may have a negative impact on the performance at initial stages.

The resultant end-to-end delay depends on the efficiency of task scheduling algorithm [51]. Figure 12.c shows the performance evaluation of different task scheduling algorithms in a fog-cloud environment, in terms of end-to-end delay. The x-axis shows the policies considered in our simulation and y-axis presents their corresponding end-to-end delay in milliseconds. The proposed LRFC policy exhibits the lowest end-to-end delay compared to FCFS, Random and SJF policies. As described earlier, the efficient task scheduling has a crucial role on the end-to-end delay. When the resources are allocated according to the job requirement keeping in view the other important factors like availability, capacity and proximity of resources (as done in LRFC), the average end-to-end time is reduced. Consequently, QoE of the system is improved. Additionally, assigning best resource for incoming job reduces the extra delays such as queuing and migration delays as well.

Finally, a comparative analysis of considered evaluation metrics is performed using 30k and 50k IoT jobs, as shown in Figure 13. It can be observed that the proposed policy exhibits a slight difference among the varied performance evaluation metrics used in the paper even with a considerable amount of increase in IoT jobs. Consequently, the system shows a normal behaviour that confirms the scalability of the proposed approach in terms of number of number of IoE jobs.

6. Conclusion and Future Work

The paper explores task scheduling thoroughly in Fog computing environments. An adaptive and intelligent task scheduling technique, Learning Repository Fog-Cloud (LRFC), has been proposed to improve QoS (i.e., response time, and processing time of tuples) and energy consumption (i.e.,
power consumption of fog devices). The authors have proposed a smart soft layer between IoE/IoT-devices and Fog nodes that can be extended to implement various types of learning based policies. The proposed deployment model exhibits scalability and thus avoid performance bottlenecks. The proposed approach has been thoroughly evaluated using extensive simulations and verified formally. The verification of the proposed approach with current state-of-the-art shows promising results both in terms of energy efficiency and QoS. Our future work includes the utilization of our proposed smart layer with various experimentation of intelligent learning based techniques in combination with varied state-of-the-art scheduling policies to mainly access varied futuristic large scale distributed computational paradigms (i.e., Edge,
(a) Average Processing Time

(b) Average Propagation Time

(c) Average End-to-end Delay

(d) Average Power Consumption Time

Figure 13: Performance evaluation of LRFC with 30k and 50k.
References


