

# Latency-aware Application Module Management for Fog Computing Environments

Redowan Mahmud, The University of Melbourne  
 Kotagiri Ramamohanarao, The University of Melbourne  
 Rajkumar Buyya, The University of Melbourne

Fog computing paradigm has drawn significant research interest as it focuses on bringing Cloud based services closer to Internet of Things (IoT) users in an efficient and timely manner. Most of the physical devices in Fog computing environment, commonly named as Fog nodes, are geographically distributed, resource constrained and heterogeneous. To fully leverage the capabilities of the Fog nodes, large scale applications that are decomposed into inter-dependent Application Modules, can be deployed orderly over the nodes based on their latency sensitivity. In this paper, we propose a latency-aware Application Module management policy for Fog environment that meets the diverse service delivery latency and amount of data signals to be processed in per unit time for different applications. The policy aims to ensure applications Quality of Service (QoS) in terms of satisfying service delivery deadline and optimizing resource usage in Fog environment. We model and evaluate our proposed policy in iFogSim-simulated Fog environment. Results of the simulation studies demonstrate meaningful improvement in performance over alternative latency-aware strategies.

CCS Concepts: •**Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; •**Networks** → Network reliability;

Additional Key Words and Phrases: Internet of Things, Fog computing, Application management, Latency-awareness, Application placement, Resource optimization, Application QoS

## ACM Reference Format:

R. Mahmud, K. Ramamohanarao, R. Buyya, 2017. Latency-aware Application Module Management in Fog Computing Environments. *ACM Trans. Embedd. Comput. Syst.* 9, 4, Article 39 (March 2010), 21 pages.  
 DOI: 0000001.0000001

## 1. INTRODUCTION

Due to rapid advancement in communication and hardware technology, it is expected that by 2020 there will exist over 50 billion Internet of Things (IoT) devices with high demand of real-time latency-sensitive applications [Vermesan and Friess 2014][Chiang and Zhang 2016]. In this context, Clouds can be used as infrastructure for hosting the IoT applications. However, as Cloud datacentres are geographically centralized in nature, it is difficult for them to support applications dealing with large number of highly distributed IoT devices. On the long run, this inconvenience will cause unacceptable high latency in service delivery and congestion in network. Therefore, to overcome these obstacles, Fog computing is introduced [Dastjerdi et al. 2016].

Fog computing refers to a hierarchically distributed computing paradigm that bridges Cloud datacentres and IoT devices. Fog computing environment offers both infrastructure and platform to run diversified software services. Hence, Fog comput-

---

Author's addresses: R. Mahmud, K. Ramamohanarao and R. Buyya, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information System, University of Melbourne, Parkville Campus, Melbourne, VIC 3010, Australia.

Email: mahmudm@student.unimelb.edu.au, kotagiri@unimelb.edu.au, rbuyya@unimelb.edu.au

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2010 ACM. 1539-9087/2010/03-ART39 \$15.00

DOI: 0000001.0000001

ing extends Cloud based facilities to the edge network, reduces service delivery latency, network congestion and improves Quality of Service(QoS) [Dastjerdi et al. 2016]. At different hierarchical levels of Fog environment, the physical devices are commonly called as Fog nodes. Traditional networking devices; like set-top boxes, gateway routers, smart switches, proxy servers etc. equipped with computational resources can be used as potential Fog nodes. Fog nodes are heterogeneous to each other in terms of resource capacity and application execution environment. Unlike Cloud datacentres, most of the Fog nodes are resource constrained due their inherent physical structure and can be deployed distributively across the edge [OpenFog Consortium 2017] [Mahmud et al. 2018]. To align large scale applications in Fog computing environment, the applications are modeled as a collection of light-weight, inter-dependent Application Modules [Taneja and Davy 2017] [Gupta et al. 2017].

Typically, different IoT applications carry out some common activities such as receiving data from IoT devices, pre-processing and analysis of the received data, handling event of interests, etc. [Gubbi et al. 2013]. An Application Module contains necessary instructions so that one of these aspects for the respective application can be attained. For a given input, an Application Module performs some specific operations to generate corresponding output. Later, based on data-dependency, the output is sent to another module as input. In order to execute properly, each module requires a certain amount of resources, e.g. CPU, memory, bandwidth, etc. Hence, Application Modules together with its allocated resources constitute the data processing elements for different applications. This sort of decomposition is effective for distributed development of large scale applications. In literature, similar concept is used to divide Component-based applications into multiple Application Components [Yangui et al. 2016].

However, while executing applications in distributed manner, latency related issues such as node to node communication delay, application service delivery deadline, and service access frequency often become predominant and influence Quality of Services (QoS) and resources utilization to a great extent. In different computing paradigms although various latency-aware management strategies for distributed applications are proposed [Kang et al. 2012] [Nishio et al. 2013] [Ottenwalder et al. 2013][Takouna et al. 2013], the aforementioned latency issues have not been addressed simultaneously. Besides, due to dependency towards centralized management and lack of latency-sensitive application prioritization, the existing policies often get interrupted to meet the challenges of IoT-enabled real-time interactions. Therefore, in this paper, we propose a latency-aware Application Module management policy for Fog computing environment that considers different latency aspects of an distributed application in a body with decentralized co-ordination. The objective of the policy is to manage latency-sensitive and latency-tolerant IoT-applications in different ways so that deadline driven QoS provision can be ensured for all types of applications along with optimizing resources in Fog computing.

The main **contributions** of this paper are:

- A latency-aware approach for placing Application Modules on distributed Fog nodes that ensures deadline satisfied service delivery for different types of applications.
- Explored latency-aware Application modules forwarding strategy that re-locates modules in order to optimize number of computationally active Fog nodes.
- Our proposed latency-aware policy is applied in iFogSim-simulated Fog environment and compared with other latency-aware policies from different perspectives. The performance results show significant improvement in favour of our policy.

The rest of the paper is organized as follows. In Section 2, we highlight several relevant research works. In Section 3, different event-driven IoT-application scenarios is discussed with a general application model. Section 4 provides the system overview,

assumptions and problem description. The proposed latency-aware Application Module management policy is presented in Section 5. Section 6 reflects the simulation environment and the performance evaluation. Finally, Section 7 concludes the paper.

## 2. RELATED WORK

[Famaey et al. 2009] propose a dynamic and latency-aware distributed service placement policy over multiple homogeneous servers. The services can be assigned with a priority value based on their utility. A server applies the policy to determine whether it can execute a request with the service placed within it or should forward the request to another service hosted in the nearby servers. While forwarding the service, server to server latency is taken into account. The proposed algorithm shuffles active services within a server so that resources can be accommodated to execute the request and latency incurred due to service forwarding can be avoided.

[Kang et al. 2012] propose an Iterative Sequential co-deployment algorithm for distributed services. Based on users proximity, the algorithm at first generates some virtual random placement for the services, and then performs iterative re-placement to improve the users service access and inter-service (nodal) communication time. The algorithm deals with both latency-sensitive and latency-tolerant services in the same way. After real-placement of the services, to handle dynamic changes of users and their access pattern, authors recommend to run the deployment process periodically.

[Takouna et al. 2013] indicate towards communication latency and energy-aware placement of parallel applications in virtualized datacentres. The proposed policy dynamically identifies the bandwidth demand and communication characteristics of the distributed applications and re-allocates the applications through migration if the current placement fails to handle the issues. A migration manager supervises the operations at the time of migration. The migration manager sorts the virtualized instances based on their current traffic and selects an appropriate instance as migration target by checking the resource availability.

[Gupta et al. 2016] propose a transfer-time aware workflow scheduling policy for multi-Cloud environment. It prioritizes inter-dependent tasks for scheduling to the multi-Clouds based on the computation cost and communication time. Within the clouds, the tasks are mapped considering earliest start and finish time. The policy aims to enhance service delivery time. The relevant operations of the proposed policy is handled by a global cloud manager.

[Fan et al. 2018] discuss data placement in geo-distributed multi-Clouds. The proposed Energy-Efficient Latency-Aware Data Deployment Algorithm (ELDD) sorts data chunks in non-ascending order according to the collective access probability from all the users and merge them into larger data segments based on the capacity of the servers. Later it iteratively searches for appropriate servers where placement cost of such data segment gets reduced. The placement cost is calculated centrally considering service access latency, energy consumption of the servers and the network. The algorithm also iterative turns-off of the free servers.

Based on the requirements during development, deployment and management of component-based IoT applications, [Yangui et al. 2016] propose a Platform as a Service (PaaS) architecture to facilitate application provisioning in hybrid Cloud-Fog environment. Being a centralized co-ordinator, the PaaS architecture allows developing applications according to the target domain; discovering, initiating, configuring and scaling resources for deploying and executing the application components; orchestrating execution flow between of the components; monitoring SLA and component migration; providing resource and component management interfaces. In evaluation, different component placement scenarios are discussed based on the end-to-end latency.

[Taneja and Davy 2017] present a Module Mapping Algorithm to place distributed applications in Cloud-Fog environment. The work aims to ensure proper resource utilization. The algorithm is aware of the network. It sorts both the nodes and application modules according to the available capacity and requirements, and then maps the modules when the constraint satisfies. In one sense, it prioritizes the modules based on the resource expectation. The policy reflects the way to reduce resource under-utilization for distributed IoT application. It also highlights how Fog-Cloud inter-operation can minimize end-to-end latency compared to traditional cloud based approach.

[Ottenwalder et al. 2013] propose a plan-based operator placement and migration policy for Mobile Complex Event Processing (MCEP) applications. It supports mobility of the users and creates time-graph model to identify possible migration targets. Considering the shortest path from data source, it selects the appropriate target instance from the time graph model. On the selected instance, the policy applies con-ordination to accommodate the migrating operator. The main intentions of the proposed policy are to reduce network overhead and end-to-end delay.

[Nishio et al. 2013] discuss latency-aware application deployment for Mobile Cloud Computing (MCC). In the developed resource sharing architecture, a coordinator manages all incoming requests and resources in order to optimize service latency for different applications. To facilitate resource sharing, smaller amount of tasks are forwarded from one node to another node under the supervision of the coordinator. The proposed policy also trade-offs utility gain and energy consumption to optimize the resources.

A cost optimized offline and online service placement policy for Mobile Micro Cloud (MMC) is discussed by [Wang et al. 2017]. The policy determines an optimal configuration of service instances that minimize the average cost over time. A centralized controller predicts the cost and computes the service instance configuration for the next time slots. The cost function can include resource consumption, service access latency and other monetary issues. The policy supports mobility of the entities and migration of the services accordingly. The authors also consider the errors while predicting the cost and develop a method to identify the optimal look-ahead window size.

[Chamola et al. 2017] consider Software Defined Network (SDN) enabled communication of multiple Cloudlets to place services at the proximity of the mobile users. The task assignment solution can improve the QoS in respect of service delivery and service access time. According to the proposed policy, if a Cloudlet gets overloaded, the tasks offloaded to it, are processed on another relaxed Cloudlet of the network. Necessary operations to conduct the policy are supervised by a central Cloudlet manager.

To facilitates latency-aware scheduling of applications in virtual machines, [Xu et al. 2012] introduces a scheduler named vSlicer. vSlicer nurture the concept of *differentiated-frequency microslicing*. Unlike, traditional scheduler, vSlicer divides a CPU slice into many microslices and according to microslices, it schedules applications in higher frequency. By doing so it increases CPU access probability of applications.

Table I provides a brief summary of state-of-the-art for latency-aware application or service placement in distributed servers, multi-Cloud, Mobile-Cloud and Cloud-Fog environment. Compared to the existing works, the unique aspect of our work is that we have considered service access delay, service delivery time and inter-nodal communication delay simultaneously while placing and forwarding inter-dependent application modules over distributed Fog nodes. Besides, the proposed policy decentrally coordinate the placement and forwarding operation to overcome the constraints of centralized supervision for example; application management overhead, single point of failure, additional communication and decision making delay etc. Our policy can place the modules both horizontally and vertically and in order to facilitate low energy usage, can optimize resources by forwarding all the modules from one node to the others. Moreover, it enhances priority of latency-sensitive applications to place closer to the

data source by deploying latency-tolerant applications in the upper level Fog nodes. Simulation results support the applicability of our policy in terms of QoS satisfaction, resource optimization application module placement and forwarding time.

Table I: Summary of the literature study

Work	Distributed application	Meets latency			Forwards application	Optimizes resources	Decentralized management	Prioritized placement
		Service access	Service delivery	Inter nodal				
[Famaey et al. 2009]	✓			✓	✓		✓	✓
[Kang et al. 2012]	✓	✓		✓			✓	
[Takouna et al. 2013]	✓			✓	✓			✓
[Gupta et al. 2016]	✓		✓	✓				✓
[Fan et al. 2018]	✓	✓				✓		✓
[Yangui et al. 2016]	✓		✓		✓	✓		
[Taneja and Davy 2017]	✓		✓			✓		✓
[Ottenwälder et al. 2013]	✓		✓		✓		✓	
[Nishio et al. 2013]	✓		✓		✓	✓		
[Wang et al. 2017]		✓			✓	✓		
[Chamola et al. 2017]		✓	✓		✓			
Latency-aware (this work)	✓	✓	✓	✓	✓	✓	✓	✓

### 3. APPLICATION SCENARIOS

#### 3.1. IoT-enabled Systems

In advanced healthcare and smart home based system, the structure of different IoT applications reflects through some common operations. Two such application scenarios and their basic operations on the received data can be described as follows.

*3.1.1. Patient Respiratory Monitoring System.* In order to monitor breathing and oxygen level of asthma patients, Pulse Oximeters are widely used at the hospitals. Usually Pulse Oximeters are connected to the bed-side monitors, and continuously shows oxygen level carried in the body, heart beat rate and changes in blood volume of the skin [Addison et al. 2015]. The bed-side monitors provide the interface for authentication, aggregate data signals and usually forward the sensed data to Cloud or other computational entities for further processing to detect Hypoxemia, Hypercapnia and sleep apnea of the patients. Since, some Pulse Oximeter generated data signals can be irrelevant, incomplete and diverse in format; data filtering techniques are applied in this context. Later, different data analytics in respect of Hypoxemia, Hypercapnia, etc. operate on the filtered data. Sometimes, outcome can indicate to an emergency situation. Based on the outcome, required actions for example ventilation, injection, medication, etc. are triggered at the patient's bed-side actuators. For critical asthma patients, corresponding application has to perform the aforementioned operations in real-time that

Cloud-based placement of the application often fails to deal with. In addition, placement of such large-scale application in distributed and heterogeneous Fog nodes is not as simple as the Cloud-based placement.

**3.1.2. Visitor Identification System.** To identify a visitor in smart-home based system, usually entrance-side cameras take the pictures of the visitors and send to Cloud or other computing entities for image processing [Sahani et al. 2015]. Sometimes due to weather condition and other external effects, the taken pictures get a lot of noises. In this context, image filtration techniques are required to apply for selecting the most appropriate picture and reducing its noises. Image analytics in respect of face and gesture recognition, object detection, etc. are also applied to the filtered pictures for identifying the visitor and the hand held objects. Once the visitor is identified and the hand held objects are found allowable, necessary information is parsed from the respective databases. The information can include contact number, address and access rights of the visitor. If the visitor is authorized to enter the house, entrance-side actuators open the door otherwise create notification to the residents containing the details of the visitor. During urgent period, corresponding application of Visitor Identification System is required to co-ordinate the aforementioned operations within a reasonable time that may not be possible if the application is placed in distant Cloud. Besides, necessary resources to execute this kind of compute intensive applications in resource constrained Fog nodes are often difficult to manage.

### 3.2. Application Model

Based on the aforementioned scenarios and data-operations, we have considered the following application model for the associate event-driven applications. we assume that each application is composed of *Client Module* (provides initial application interface), *Data Filtering Module* (applies data filtering techniques), *Data Analysis Module* (executes data analytics) and *Event Handler Module* (generates appropriate response to the event). Data dependency exists among the modules of same application which can be expressed through a sequential unidirectional data flow as shown in Fig. 1. After placement of an Application Module, from the respective data flow the next module is identified for placement. To foster concurrency, the modules can be replicated. Besides, if a module is allocated resources according to its requirements, it is expected that the module will execute its operation within a fixed time.

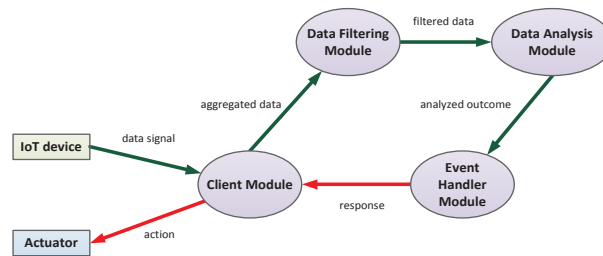


Fig. 1: Application Model

Client Module is the entrance module for each application. Application initializing information such as authentication, data signal sensing frequency, service delivery deadline, metadata of subsequent Application Modules and their inter-dependency are notified to the system through the Client Module. After deployment of all modules, Client Module of a particular application directly communicates with respective IoT



devices to grab their sensed data signals and forwards in form of aggregated data to the subsequent modules for further operations. If the ultimate analyzed outcome of a forwarded data signal invokes any event of interest, Event Handler Module sends corresponding response towards the Client Module. This response is eventually considered as the final application service for the generated data signal. Based on the response, Client Module triggers action the actuators.

Since *Client Module* plays the role of root module for the applications and closely associates with the IoT devices and the actuators, this particular module for every application is expected to be placed at the lower level Fog nodes.

## 4. SYSTEM MODEL AND ASSUMPTIONS

### 4.1. Organization of Fog Layer

In this work, Cloud is considered as a stand alone computational platform and IoT devices only generate data signals without further processing due to resource and energy constraints. In this circumstance, Fog computing acts as an intermediate layer in between Cloud and IoT devices. Within this layer, nodes are organized in a hierarchical order as shown in Fig. 2. Lower level Fog nodes are closer to the IoT devices. As the level number goes higher, distance of Fog nodes from IoT devices increases which can be reflected in lower level to higher level uplink latency and delay in service delivery. Compute, storage and networking capabilities of lower level Fog nodes is less compared to that of higher level nodes [Ashrafi et al. 2018]. Each node in a particular level is directly associated with a node of immediate upper level. Fog nodes can form clusters among themselves and rapidly communicate with each other through Constrained Application Protocol (CoAP), Simple Network Management Protocol (SNMP), etc. [Slabicki and Grochla 2016]. Therefore, maximum nodal communication delay  $\epsilon_C$  within a Fog cluster  $C$  is negligible and does not impact on service delivery time extensively. We assume that at lower Fog levels, if two nodes from the same level are connected with identical uplink node and experience approximately equal amount of uplink latency, the nodes can belong to the same cluster. Besides, in a reliable IoT-enabled system, it is expected that the Fog infrastructure providers have applied efficient networking techniques to ensure persistent communication among the nodes through less variable inter-nodal latency [Kempf et al. 2011].

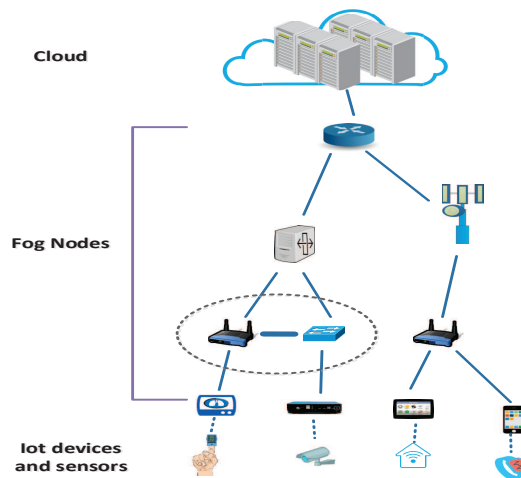


Fig. 2: Organization of Fog

The relevant notations and definitions used in modeling the system are listed in Table II.

Table II: Notations

Symbol	Definition
$N$	Set of all Fog nodes.
$A$	Set of all Applications.
$M$	Set of all Application Modules.
$R$	Set of all resources (e.g. CPU, memory, bandwidth, etc.).
$M_a$	Set of all Application Modules belongs to application $a \in A$ ; $M_a \subset M$ .
$M_n$	Set of all Application Modules placed in node $n \in N$ ; $M_n \subset M$ .
$\gamma_a^{m'm}$	Estimated inter-module data dependency delay of module $m$ from its previous module $m'$ on the data flow of application $a$ ; $m', m \in M_a$
$\delta_a^{m'm}$	Tolerable inter-module data dependency delay of module $m$ from its previous module $m'$ on the data flow of application $a$ ; $m', m \in M_a$
$\Delta_{n'n}$	inter-nodal communication delay between node $n'$ and $n$ ; $n, n' \in N$ .
$f_m^m$	Input receiving frequency of module $m$ ; $m \in M$ .
$T_{n'n}^m$	Required time to forward module $m \in M$ from node $n'$ to $n$ ; $n', n \in N$ .
$\phi_n^m$	Input processing time of module $m \in M$ in node $n \in N$
$r_{req}^m$	Requirement of resource $r \in R$ for module $m \in M$ .
$\psi_{nr}$	Capacity of node $n \in N$ for resource $r \in R$ .
$r_{avail}^n$	Available resource $r \in R$ in node $n \in N$ .
$C_n$	Cluster to which node $n \in N$ belongs.
$\epsilon_C$	Maximum communication delay within Cluster $C$ of Fog nodes
$t^{now}$	Current timestamp.
$t_m^{last}$	Timestamp when module $m \in M$ received last input.
$\mu_n^m$	Assigned MCI to module $m \in M$ in node $n \in N$ .
$y_n \in \{0, 1\}$	Equals to 1 if node $n \in N$ is computationally active, 0 otherwise.
$x_{mn} \in \{0, 1\}$	Equals to 1 if module $m \in M$ is mapped to node $n \in N$ , 0 otherwise.
$x'_{mn} \in \{0, 1\}$	Equals to 1 if module $m \in M$ was earlier deployed in node $n \in N$ , 0 otherwise.

#### 4.2. Fog node architecture

In order to define the architecture of Fog nodes for distributed application management, we have extended the concept of OpenFog consortium proposed Fog reference architecture [OpenFog Consortium 2017]. We assume, a Fog node is composed of three main components; *Controller Component*, *Computational Component* and *Communication Component* as depicted in Fig. 3.

Computational Component provides necessary resources to execute Application Modules. Inside Computational Component, modules are assigned to *Micro Computing Instances*, *MCI* where resources e.g. CPU, memory, bandwidth, etc. are allocated according to the requirements. Due to resource constraints, a Fog node can configure certain number of MCIs at a time and the configured MCIs work independently.

In a Fog node when no MCIs are running, its Computational Component is turned off. In this case, the node only serves traditional networking functionalities like routing, packet forwarding, etc. through its Communication Component. If the load of applications increases in the Fog layer, Computational Component of that node can be turned on again to handle the event. Controller Component of a Fog node monitors and manages the operations of Computational and Communication Component.

Moreover, Controller Component maintains several data structures. Among them, the *Module Sleeping Block (MSB)* contains non-executing Application Modules. When an Application Module has no input to process, it is withdrawn from the assigned MCI and placed to the MSB. Application Modules forwarded from another node also reside inactively in the MSB while they are waiting for scheduling. In addition, a Fog node tracks the Application Modules that are deployed within it using the *Placement List (PL)* and stores route related information of other Application Modules in its *Routing*



*List (RL)*. After the execution of a module, to determine host (both node and MCI) of the next module, either PL or RL of corresponding Fog node is referred to. Besides, a Fog node preserves meta-data of the placed modules even when they are no longer associated with it and the context information of other nodes in a *Data container*.

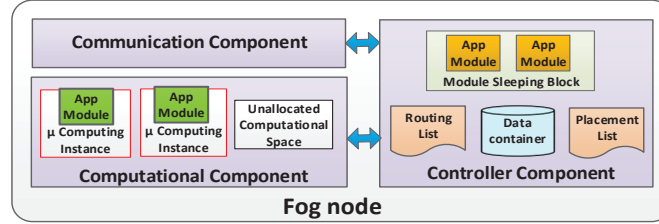


Fig. 3: Fog node components

### 4.3. Latency Model

Due to data-dependency, generated output of an Application Module is sent to another module as input. The tolerable inter-module data dependency delay  $\delta_a^{m'm}$  of module  $m$  in application  $a$  refers to the maximum amount of time that the module can wait without affecting the application's service delivery deadline to get the input from the previous module  $m'$  for a particular data signal. For any application  $a$ , the service delivery deadline can be set according to  $\sum \delta_a^{m'm}; \forall m \in M_a$ . Here, we assume  $\delta_a^{ClientFilter} + \delta_a^{FilterAnalysis} + \delta_a^{AnalysisEvent} < \delta_a^{EventClient}$ . As a consequence, tolerable inter-module data dependency delay-aware placement of Data Filtering, Analysis and Event Handler module spontaneously justify the placement of the Client module.

However, inter-module data dependency delay of a module  $m$  placed in node  $n$  can be estimated ( $\gamma_a^{m'm}$ ) based on the input processing time  $\phi_n^{m'}$  of the previous module  $m'$  placed in node  $n'$  and the inter-nodal communication delay  $\Delta_{n'n}$  between their host nodes. In some cases, input scheduling delay of the previous module can also contribute to the estimated inter-module data dependency delay of the respective module.

In addition, the service access rate of data signals for an application and replication number of previous module play an important role to the input receiving frequency of a particular Application module. The input receiving frequency  $f_m$  of a module  $m$  itself helps to identify the possible idle period of module  $m$ . For example, if  $f_m = 2/ms$  and  $\phi_n^m = 0.2$  ms in node  $n$ , then it can be expected that after processing an input, module  $m$  in node  $n$  will be remain idle for the next 0.3 ms. Within this idle period of module  $m$ , its assigned MCI  $\mu_n^m$  can be allocated to other module for input processing.

### 4.4. Module Management Problem

Usually lower level Fog nodes are not resource enriched like upper level nodes although placement of applications on lower level nodes facilitate faster service access and service delivery. Besides, not all the applications show identical responses to latency related issues. For latency-sensitive applications, service delivery deadline so as to module's tolerable data dependency delay is stringent compared to the latency-tolerant applications. In this case, placement of latency-tolerant applications in limited number of lower level Fog nodes can obstruct many latency-sensitive applications to meet their requirements. Conversely, by considering lower Fog level scalable, if all applications are placed there, upper-level nodes will remain under-utilized. Therefore, an efficient module placement policy is required that can prioritize applications to place

in closer proximity of data source by taking necessary latency-related issues into account. More precisely, the policy should identify which applications (modules) should be placed at lower Fog level and which are required to move towards upper Fog level.

Moreover, in some cases, the number of computationally active Fog nodes is needed to optimize so that energy usage, expenses, etc. get minimized. In this case, some modules are required to forward from one node to another. The selection of source and destination node for such module forwarding is very crucial. In addition, while forwarding modules, constraints on nodes capacity, service delivery deadline and forwarding cost (e.g forwarding time) should be observed simultaneously.

In distributed environment like Fog, if the decisions regarding application management is taken decentrally, both application placement time and overhead from the centralized controller will be reduced. The application management can be done without relying on a single entity although it will be very difficult to co-ordinate the nodes.

## 5. PROPOSED APPLICATION MODULE MANAGEMENT POLICY

Our proposed latency-aware Application Module management policy runs on the Controller Component of each Fog nodes without supervision of any external entity. This management policy basically targets application module placement to ensure deadline satisfied QoS and resource optimization in the Fog layer.

### 5.1. Assurance of QoS

To initiate any application  $a$  in Fog, the corresponding IoT devices subscribe with a Fog node. This node acts as the *Application gateway node* for application  $a$ . Usually during subscription, the Client Module of application  $a$  is by default placed on its Application gateway node. Therefore, Application gateway nodes of different applications is located at the lower Fog level. However, placement of the module next to Client Module also initiates from Application gateway node. In order to initiate placement of a module, Fog node executes the *PlaceAppModules* procedure given in Algorithm 1.

*PlaceAppModules* procedure takes the to be placed Application Module,  $m$ , its previous module,  $m'$  and observed network delay,  $\omega$  as arguments.

As shown in Algorithm 1, *PlaceAppModules* procedure basically consists of 4 steps: At first, the procedure inquires context of the current node (line 2). If the current node is Cloud, rest unassigned modules will be placed there, otherwise the procedure inquires context of the corresponding uplink node and host node of  $m'$  (line 3-7). Then the following steps are executed:

- 1) Sum of the input processing time of previous module  $m'$ , observed network delay from host node of  $m'$  to the current node and the current node's uplink latency is checked with tolerable inter-module data dependency delay of the to be placed Application Module,  $m$  (line 9). If it is feasible to route module  $m$  to the uplink node, the current node updates its RL for the module. At the uplink node, deployment process of the module is re-initiated by invoking its *PlaceAppModules* procedure with an updated value of observed network delay,  $\omega$  (line 10-14).

- 2) If it is not efficient to route module  $m$  to the uplink node, the current node intends to place the module within itself. In order to do so, the resource availability of the current node is checked with the requirement of module  $m$ . If the resource availability supports requirements of module  $m$ , the current node update its PL for module  $m$  (line 15-16). However, as the module is deployed in a computationally active node, boolean variable  $\eta$  is set to *true* (line 17) and availability of the resources in the current node is updated (line 18).

- 3) Another computationally active node from the same cluster as the current node is selected to place the module  $m$  if available resources at the current node do not meet the module's requirements. This selection is also conducted based on the resource

**Algorithm 1** Module placement algorithm

---

```

1: procedure PLACEAPPMODULES( $m, m', \omega$ )
2:    $p \leftarrow$  this node
3:   if  $p = \text{Cloud}$  then
4:     place rest modules in cloud
5:     return
6:    $q \leftarrow p.\text{uplinkNode}$ 
7:    $z \leftarrow m'.\text{hostNode}$ 
8:   if  $m \neq \text{null}$  then
9:     if  $\phi_z^{m'} + \Delta_{pq} + \omega < \delta_a^{m'}$  then
10:       $\omega \leftarrow \omega + \Delta_{pq}$ 
11:       $p.\text{RL.add}(m, q)$ 
12:       $q.\text{PlaceAppModules}(m, m', \omega)$ 
13:     else
14:       $\eta \leftarrow \text{false}$ 
15:      if  $r_{req}^m < r_{avail}^p, \forall r \in R$  then
16:         $p.\text{PL.add}(m)$ 
17:         $\eta \leftarrow \text{true}$ 
18:         $p.\text{update}(r_{avail}^p)$ 
19:      else
20:        for  $u := C_p.\text{activeNodes}$  do
21:          if  $r_{req}^m < r_{avail}^u, \forall r \in R$  then
22:             $p.\text{RL.add}(m, u)$ 
23:             $u.\text{PL.add}(m)$ 
24:             $\eta \leftarrow \text{true}$ 
25:             $u.\text{update}(r_{avail}^u)$ 
26:          break
27:      if  $\eta = \text{false}$  then
28:        select node  $v \in C_p.\text{inactiveNodes}$ 
29:         $p.\text{RL.add}(m, v)$ 
30:         $v.\text{PL.add}(m)$ 
31:         $v.\text{update}(r_{avail}^v)$ 
32:       $m' \leftarrow m$ 
33:       $m \leftarrow m'.\text{getNext}$ 
34:       $p.\text{DeployAppModules}(m, m', \epsilon_{C_p})$ 
35:   else
36:     return

```

---

availability of other cluster nodes. In this case, current node updates its RL and the selected cluster node updates its PL, resource availability for module  $m$  (line 20-25).

4) If all computationally active cluster nodes fail to allocate resources for deploying module  $m$ , an arbitrary computationally inactive node from the cluster will be selected to place the module. RL and PL of the respective nodes will be updated for the module (line 27-31).

Step 2-4 of the algorithm operate on the same cluster. Therefore, placement process of the next Application Module can be initiated from any node of the cluster. In this algorithm, current node is selected to do so as it simplifies management of routing information. Since, Fog nodes residing in same cluster are connected with faster networking protocols (e.g. CoAP, SNMP, etc.), observed network delay  $\omega$  in this case is considered negligible and set equal to  $\epsilon$  of the Cluster.

Algorithm 1 can be easily extended to handle the scenario when there exist no inactive nodes to host an application module within a cluster. In this case, the module can be bypassed to the proximate clusters provided that the tolerable inter module data-dependency delay is not violated. If still the module is failed to deploy, it can be sent either to the uplink nodes or to the proximate cluster nodes where tolerable inter module data-dependency delay gets less violated. It is done so that even if the deadline

cannot be meet, service delivery time remain as low as possible. However, if the Fog computing platform is unable to allocate resources for all the modules of an application, it notifies the user through application gateway node to flexible the deadline so that it can be placed to the Cloud.

In a reliable IoT-system where the requirements of modules assist them to process input within a fixed amount of time, The proposed Algorithm 1 helps the applications to meet their service delivery deadline. In implicitly deals with latency-sensitive and tolerant applications in different way. According to the policy Latency-tolerant applications (modules) are placed vertically whereas latency-sensitive applications (modules) are placed horizontally across the cluster and in lower Fog level resources are preserved for future latency-sensitive applications.

## 5.2. Optimization of Resources

Generally, if all deployed Application Modules of a particular Fog node are re-located to other nodes for further execution, Computational Component of that node can be turned off. Hence, the number of computationally active Fog nodes can be reduced. In Fog, this sort of optimization can be handled in terms of both Linear Programming and heuristic based approaches.

*5.2.1. Formulation of a Linear Programming problem.* A constrained Linear Programming (LP) problem is formulated in Eq. 1 to minimize the number of computationally active Fog nodes. It helps to identify possible target Fog node  $n$  for re-locating Application Module  $m$  through a binary decision variable  $x_{mn}$ . Another binary variable  $x'_{mn}$  tracks whether module  $m$  has been available in node  $n$  since the last placement. The constraints ensures that a module will not be mapped to multiple nodes (Eq. 2), resources of the target node satisfy the module's requirements (Eq. 3), placement of the module to target node does not affect the tolerable inter-module data dependency delay of the next module (Eq. 4) and the required time to forward the module from source node to target node fits within the input arrival interval of that module (Eq. 5).

$$\min \sum_{n \in N} y_n \quad (1)$$

subject to,

$$\sum_{n \in N} x_{mn} = 1; \forall m \in M \quad (2)$$

$$\sum_{m \in M} r_m^{req} x_{mn} \leq \psi_{nr} y_n; \forall n \in N, \forall r \in R \quad (3)$$

$$x_{mn} \gamma_a^{mm''} \leq \delta_a^{mm''}; \forall n \in N, \forall a \in A, \forall m \in M_a \quad (4)$$

$$T_{n'n}^m (x_{mn} - x'_{mn}) \leq \frac{1}{f_m}; \forall n, n' \in N, \forall m \in M \quad (5)$$

This LP problem is required to be solved periodically to optimize the number of computationally active nodes. Any integer programming solver e.g SCIP [Achterberg 2009] can be used in this case. However, based the solution of LP problem, modules can be re-located in optimal number of nodes and Computational Component of other active nodes can be turned off.

*5.2.2. Proposed heuristic solution.* In a Fog environment with large number of computationally active nodes, the aforementioned LP problem takes much time to be solved. As a consequence, in making real-time forwarding decisions the LP-based solution will not be acceptable. Therefore, here we propose a heuristic based solution to the problem.

In the heuristic approach, we consider that after latency-aware Application Module placement of any application  $a$ , Fog nodes belonging to the same cluster share their context (e.g. PL, RL, Data container information, etc.) with each other. This sort of context sharing among the nodes is conducted within  $T_s$  amount of time which is termed as *context sharing period*. After context sharing period, different Fog nodes are found hosting different number of Application Modules. Based on a predefined threshold percentage of allocated resources, some nodes are identified as highly-occupied while others are considered under-occupied.

Due to step 2 and 3 of Algorithm 1, the number of under-occupied nodes in a cluster is comparatively less than highly-occupied nodes. Moreover, to make an under occupied node computationally inactive, only a few Application Modules will be required to re-locate. For example, let us assume, there is a cluster of four nodes, each of them can host upto three Application Modules with similar resource requirements. At a particular time, two of them are occupied with two modules each and rest are occupied with one module. The resource allocation threshold for each node is set to 60%. In this case re-location of two Application Modules from a highly-occupied node to other nodes make only one node computationally inactive whereas re-location of Application Modules from two under-occupied nodes can make two nodes computationally inactive. Taking this concept into account, the proposed heuristic approach aims at re-locating modules from under-occupied nodes to other highly-occupied cluster nodes.

In order to conduct re-location of Application Modules from an under-occupied node,  $n_u$  to other highly-occupied cluster nodes, at first  $n_u$  forwards the modules in non-executing form to each of the nodes. Within the highly-occupied cluster nodes, forwarded modules reside in MSB. If a highly-occupied node accommodates any of the forwarded modules in its Computational Component, the RL of  $n_u$  is updated for that module. In this case, other cluster nodes discard the module from their MSB. Otherwise, at highly-occupied cluster nodes, forwarded modules are required to be scheduled in MCIs of other Application Modules.

After forwarding non-executing form of all placed modules, an under-occupied node  $n_u$  usually tries not to execute the modules in its Computational Component. In this case, if  $n_u$  receives input  $\tau$  for module  $m$  of application  $a$ , it either routes the input to new host node of the respective module or asks a suitable highly-occupied node to schedule the module through *ForwardAppModules* procedure (Algorithm 2).

Algorithm 2 is consisted of two basic steps. After finding the context of current node (line 2), the following steps are executed:

1) In the RL of current node, if reference of new host node for module  $m$  is found, input  $\tau$  will be sent to that node (line 3-6).

2) To identify a suitable host module and its assigned MCI for scheduling Application Module  $m$ , Algorithm 2 takes each highly occupied cluster nodes into account (line 10-11), parse relevant information (line 12) and checks the following conditions:

*i.* host module is not currently processing any input (line 13). It ensures that,  $m$  will be scheduled in MCI of host module only when it is idle.

*ii.* host module will not receive any input until module  $m$  finishes input processing (line 14). This condition ensures that re-location process will not discard any input of host module.

*iii.* the assigned MCI to host module meets the resource requirements of module  $m$ (line 15).

**Algorithm 2** Module forwarding algorithm

---

```

1: procedure FORWARDAPPMODULES( $m, a, \tau$ )
2:    $p \leftarrow$  this node
3:    $q \leftarrow p.RL.get(m)$ 
4:   if  $q \neq null$  then
5:     send  $\tau$  to  $m$  on node  $q$ 
6:   return
7:    $m'' \leftarrow m.getNext$ 
8:    $host_n \leftarrow null$ 
9:    $host_m \leftarrow null$ 
10:  for  $n' := C_p.highNodes$  do
11:    for  $m' := M_{n'}$  do
12:       $n'.getInfo(m')$ 
13:      if  $t^{now} > t_{m'}^{last} + \phi_{n'}^{m'}$  then
14:        if  $t_{m'}^{last} + \frac{1}{f_{m'}} - t^{now} > \phi_{n'}^m$  then
15:          if  $r_{req}^{m'} \geq r_{req}^m, \forall r \in R$  then
16:            if  $x_{mm'} = 1 \& \gamma_a^{mm''} \leq \delta_a^{mm''}$  then
17:              if  $\lambda_{m'} = false$  then
18:                 $host_n \leftarrow n'$ 
19:                 $host_m \leftarrow m'$ 
20:                 $\lambda_{m'} \leftarrow true$ 
21:                 $n'.updateInfo(m')$ 
22:              break
23:            if  $host_n \ \&\& \ host_m \neq null$  then
24:              break
25:            if  $host_n \ \&\& \ host_m \neq null$  then
26:              schedule  $m$  in  $\mu_{host_m}$  on node  $host_n$ 
27:              send  $\tau$  to  $m$  on node  $host_n$ 

```

---

iv. Placement and execution of module  $m$  on the host node will not affect the tolerable inter-module data dependency delay of its next module(line 16).

v. no other under-occupied nodes have selected the MCI assigned to host module for scheduling their Application Modules (line 17). For any host module,  $m'$  this condition is observed through boolean variable  $\lambda_{m'}$ . **After identifying the host node, necessary information are updated (line 18-21)**. As soon as scheduled Application Module finishes input processing,  $\lambda_{m'}$  is set to *false* again.

Here, Algorithm 2 employs first fit solution to schedule Application Module  $m_A^i$ . However, after re-location of modules from under-occupied nodes to highly-occupied nodes, there will be an observation period. Within this period if no anomaly (e.g. failure in scheduling forwarded modules, QoS degradation, etc.) is detected, soon after the observation period, Computational Component of under occupied nodes will be turned off. Hence, the number of computationally active nodes from the Fog can be reduced. **Besides, the proposed policy dynamically determines host node and host module for the forwarded modules which helps to deal with sudden changes in input receiving frequency (e.g. due to add new replica) of the modules.**

Our proposed heuristic based resource optimization through latency-aware Application Module forwarding operates within clusters. In this approach, usually a small number of Application Modules from under-occupied nodes are forwarded to highly-occupied cluster nodes. Since highly-occupied cluster nodes contain many potential host modules, there will be always a possibility of finding suitable MCI to schedule less amount of forwarded modules. Moreover, cluster nodes are connected with each other with faster communication protocols. Therefore, communication latency during module forwarding is negligible and does not obstruct application QoS. **However, for a forwarded module if no suitable host node and host module is found, the module**



will be executed in its initial placement. In that case, there will be no further scope of forwarding data signal endlessly without being accepted.

## 6. PERFORMANCE EVALUATION

The performance of the proposed Application Management policy is evaluated in two phases; At first, the proposed latency-aware module placement is compared with the approaches mentioned in [Kang et al. 2012] and [Nishio et al. 2013]. In [Kang et al. 2012], a latency-aware iterative algorithm is introduced to place applications whereas in [Nishio et al. 2013], a centralized resource coordinator supervised service oriented resource sharing (SORS) is discussed. In this phase, deployment time of modules, percentage of deadline satisfied data signals are considered as the performance metrics.

Later, the proposed heuristic based solution for resource optimization is compared with the solution of LP problem. In solving the LP problem, SCIP solver [Achterberg 2009] is used. The proposed latency-aware Application Module forwarding is also compared with MigCEP [Ottenwalder et al. 2013] and Peer VMs Aggregation (PVA) [Takouna et al. 2013]. In MigCEP, to forward applications time-graph models are generated, algorithms for shortest path and co-ordination are executed whereas in PVA, a migration manager handles necessary steps to forward applications. In this phase, number of reduced Fog nodes, required time for identifying the target nodes and forwarding the modules are considered as performance metrics. Moreover, when scheduling of forwarded modules are required in the target nodes, the performance of the proposed approach in reducing the number of context switching is compared with vSlicer [Xu et al. 2012] and earliest start time-based scheduling [Gupta et al. 2016]. Increasing number of context switching can incur high service waiting time and cost.

In addition, the performance of the proposed policy is discussed in terms of varying application contexts such as variable input processing and communication time of the modules along with sudden changes in application service access rate.

### 6.1. Simulation Environment

To evaluate the performance of the proposed policy, a Fog environment is simulated in iFogSim [Gupta et al. 2017]. iFogSim is built upon CloudSim [Calheiros et al. 2011] framework that is used widely for simulating different computing paradigms [Lin et al. 2014] [Mahmud et al. 2016]. The simulation parameters are summarized in Table III.

In the modeled environment, we assume that Fog layer consist of three levels e.g. lower level (*LL*), mid level (*ML*), higher level (*HL*) and every node is heterogeneous to each other in terms of resource capacity and application execution environment. To

Table III: Simulation parameters

Parameter	Value
Simulation Duration	120- 240sec
Status sharing and observation period	10sec
Uplink latency:	
IoT device to LL nodes	10-15 ms
LL nodes to ML nodes	30-40 ms
ML nodes to HL nodes	60-80 ms
HL nodes to Cloud	140-160 ms
Processing time:	
Client Module	20-40 ms
Filter Module	10-20 ms
Analysis Module	150-200 ms
Event handler Module	20-40 ms
Applications service delivery deadline	350 - 750 ms
Delay to connect with centralized manager at ML	45-60 ms
Maximum nodal communication delay within Fog cluster	3-5 ms
Applications data receiving frequency	3/sec - 7/sec

conduct the experiments, we have used synthetic workload as compatible real workload for the proposed Application Management policy is not currently available. The value of simulation parameters within a specific range is determined by a pseudo random number generator. Here, application initiation request can be originated from any location at any time. We consider that due to incompleteness of data, deployed applications discard 2-3% of received signals during data filtration and 65% of the placed applications are comparatively more latency-sensitive than the rest.

## 6.2. Performance in Application Module deployment

In Iterative algorithm, at first modules are deployed temporarily in different nodes. Then, for reducing service latency, modules are gradually re-located to suitable nodes through iterations. As the application number increases, required time for iteration also gets high. In SORS policy, to place modules, each time resource coordinator is required to be asked for suitable nodes. In the proposed module placement approach, neither iteration nor supervised resource discovery is applied. Therefore, to place increasing number of applications, the proposed requires less amount deployment time compared to others (Fig. 4). This experiment also reflect that application placement decisions taken centrally can linger the placement of the applications in distributed Fog environment.

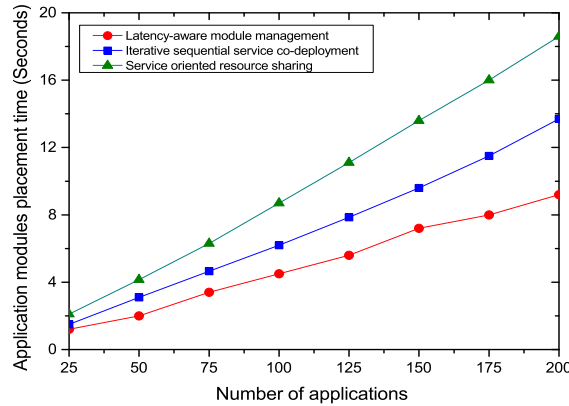


Fig. 4: Application Module deployment time vs number of applications.

Fig. 5 depicts the percentage of deadline satisfied data signals for increasing number of applications. Iterative algorithm treats both latency-sensitive and tolerant applications in a similar way. As a result, in some cases, percentage of deadline satisfied data signals for sensitive applications degrades. In SORS, for sending input to each modules of an application, resource coordinator is sent request for finding the host nodes. Additional time is required to conduct this operation which adversely affects the percentage of deadline satisfied data signals. In our proposed approach, host nodes send input from one module to another and due to place modules based on latency-endurance, neither latency sensitive nor tolerant applications are penalized in meeting deadline for processing the received data signals. This experiment result indicate that when in a system diversified applications in respect of latency-endurance exist, it is always a good policy to handle them separately.

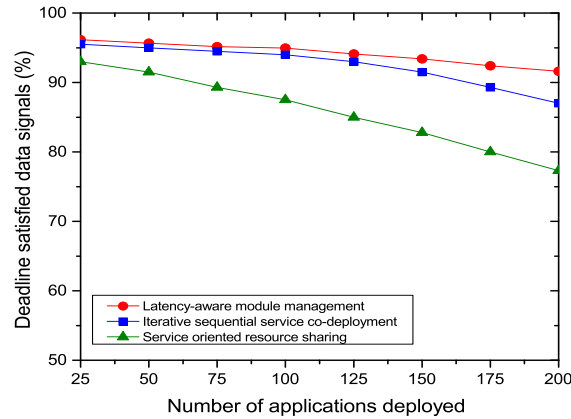


Fig. 5: Percentage of QoS satisfied data signals vs total generated data signals (input receiving frequency of applications does not change).

**6.3. Performance in Application Module forwarding**

Fig. 6 shows the comparison of LP based solution and the proposed heuristic solution in optimizing the number of computationally active Fog nodes. From the experimental results, it is found that the proposed heuristic solution is very much closer to the optimal. In this experiment, every after 10 seconds, the LP problem has been solved.

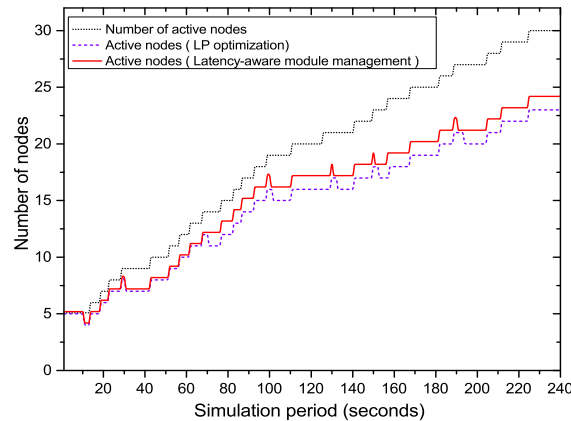


Fig. 6: Comparison of LP and heuristic based solution

In Fig. 7, required time to identify possible target nodes for module forwarding is depicted for both LP and heuristic based solution. The heuristic based solution can find suitable target nodes within a cluster by executing a simple threshold comparison. When all clusters in Fog applies the heuristic approach in contemporary basis, less amount of time will be required to identify possible target nodes from the whole system. However, in LP based solution, required time for identifying target nodes exponentially increases as the number of nodes increases. [This experiment result defines that in a system where real-time interactions happen very frequently, solving a time consuming LP problem for forwarding modules is not very efficient .](#)

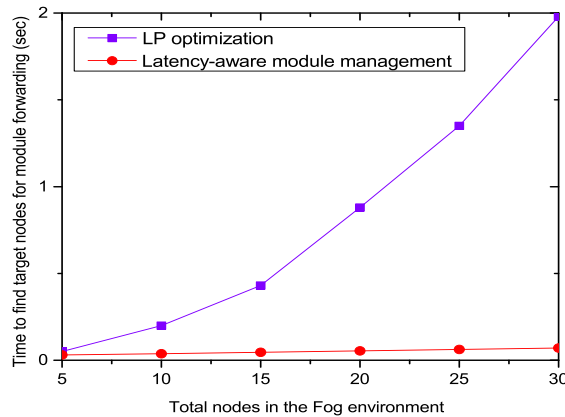


Fig. 7: Required time for generating LP and heuristic solution

A comparative study of the proposed module forwarding approach, MigCEP and PVA is depicted in Fig. 8. In MigCEP, several operations such as time-graph model generation, shortest path identification and co-ordination are conducted to forward modules. In PVA, identification of target nodes, competence checking and communication management during module forwarding are observed by a migration manager. Due to aforementioned reasons, both approaches require higher amount of time. In the proposed approach, rather than identifying a suitable node, modules are forwarded to every competent nodes in the cluster. As cluster nodes are connected with each other through faster networking standard, this type of module forwarding requires less amount of time compared to others. Although it brings additional cost for storage, for management of real time applications, it can be overlooked. Besides, this experiment signifies that formation of high-speed cluster among Fog nodes can contribute extensively to forward modules so as to optimize resources.

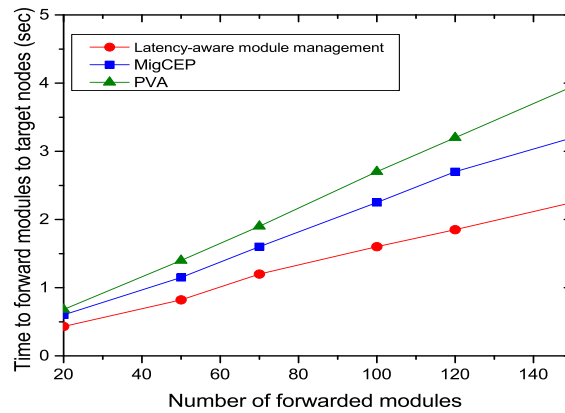


Fig. 8: Required time for Application Module forwarding vs number of forwarded modules

Fig. 9 depicts how input receiving frequency of host module influences context switching when a forwarded module is scheduled in host module's MCI. In the pro-

posed approach, if the host module’s frequency increases, number of context switching decreases whereas in *vSlicer* scheduler, this number remains the same (here, 16 context switching per second) and for early arrival time-based scheduling it increases. Rapid context switching increases overhead and waiting time at the host node node. In the proposed approach, a forwarded module only get access to the host module’s MCI when the module is idle. Therefore no data signal of both host and scheduled module waits for long time and additional overhead of context switching is reduced. **This experiment highlights that module forwarding decisions in distributed Fog environment should be taken dynamically based on the context of the nodes.**

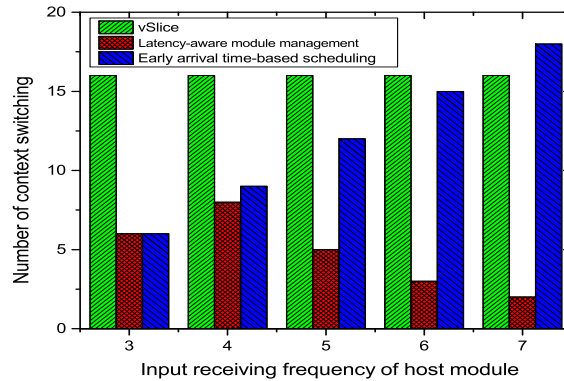


Fig. 9: Number of context switching vs frequency of host module

Fig. 10 represents how the proposed policy deals with varying application context. The application context can be varied in terms of input processing and communication time of the modules and the service access rate of the data signals. The experiment result shows that, if the processing time of the modules varies with course of time for most of the applications, percentage of QoS-satisfied data signals will be reduced. However for varying inter-communication among the modules (inter-nodal communication latency), this QoS degradation rate is higher compared to processing time variations of the modules since in distributed placement, it is considered as the dominant factor.

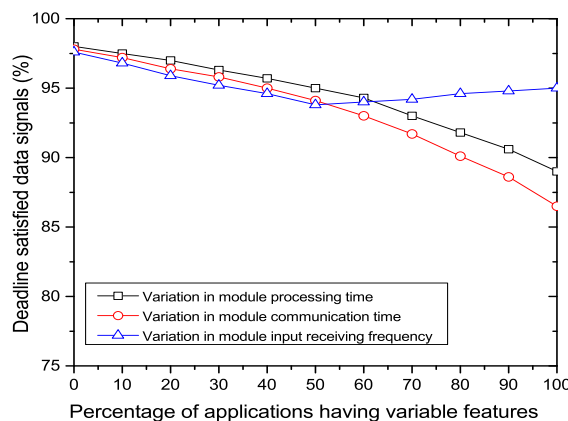


Fig. 10: Performance of the proposed policy for varying application characteristics

Moreover, if the service access rate of the applications so as to the modules changes dynamically, initially QoS-degrade specially for the forwarded modules. When the percentage of varying applications gets increased, according to the policy, no modules are forwarded. As a consequence, QoS-satisfaction rate increases. The experiment is conducted by varying one parameter at a time and the results signifies that the proposed policy works well for reliable IoT enabled system where inter-nodal communication delay does not vary significantly and all the modules are allocated with resources according to their requirements.

## 7. CONCLUSIONS AND FUTURE WORK

The Fog computing paradigm has a great potential to support a wide variety of IoT applications. We propose a latency-aware Application Module management policy that targets both deadline based QoS of applications and resource optimization. The proposed management policy meets the latency in service delivery for applications having rigorous deadline. Besides, it investigates how to optimize number of resources without violating QoS of the applications. Two algorithms have been developed in support of our proposed application management policy. The first is about Application Module placement and the second one simplifies a constrained based optimization problem in forwarding modules towards the inactive resources of idle modules. We also conducted simulation experiments in iFogSim, which shows the potential of the proposed policy.

However, we plan to implement our proposed policy in real-world. Besides, we target to place Application Modules in Fog according to users customized settings and mobility. Non deterministic latency-aware application module placement and run-time requirement adjustment of the modules can also be explored in future.

## ACKNOWLEDGMENTS

The authors would like to thank Chenhao Qu, Kyong Hoon Kim and Maria Salama for discussions and comments on improving the paper.

## REFERENCES

- Tobias Achterberg. 2009. SCIP: solving constraint integer programs. *Mathematical Programming Computation* 1, 1 (2009), 1–41.
- Paul S. Addison, James N. Watson, Michael L. Mestek, James P. Ochs, Alberto A. Uribe, and Sergio D. Bergese. 2015. Pulse oximetry-derived respiratory rate in general care floor patients. *Journal of Clinical Monitoring and Computing* 29, 1 (01 Feb 2015), 113–120.
- Tasnia H. Ashrafi, Md. Arshad Hossain, Sayed E. Arefin, Kowshik D. J. Das, and Amitabha Chakrabarty. 2018. *IoT Infrastructure: Fog Computing Surpasses Cloud Computing*. Springer Singapore, Singapore, 43–55.
- Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. 2011. 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.
- Vinay Chamola, Chen-Khong Tham, and GS S Chalapathi. 2017. Latency aware mobile task assignment and load balancing for edge cloudlets. In *Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017 IEEE International Conference on. IEEE, 587–592.
- M. Chiang and T. Zhang. 2016. Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal* PP, 99 (2016), 1–1.
- A.V. Dastjerdi, H. Gupta, R.N. Calheiros, S.K. Ghosh, and R. Buyya. 2016. Chapter 4 - Fog Computing: principles, architectures, and applications. In *Internet of Things: Principles and Paradigms*, Rajkumar Buyya and Amir Vahid Dastjerdi (Eds.). Morgan Kaufmann, 61 – 75.
- Jeroen Famaey, Wouter De Cock, Tim Wauters, Filip De Turck, Bart Dhoedt, and Piet Demeester. 2009. A latency-aware algorithm for dynamic service placement in large-scale overlays. In *Integrated Network Management, 2009. IM'09. IFIP/IEEE International Symposium on*. IEEE, 414–421.
- Yuqi Fan, Jie Chen, Lusheng Wang, and Zongze Cao. 2018. *Energy-Efficient and Latency-Aware Data Placement for Geo-Distributed Cloud Data Centers*. Springer International Publishing, Cham, 465–474.



- Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 29, 7 (2013), 1645–1660.
- Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. 2017. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience* 47, 9 (2017), 1275–1296.
- Indrajeet Gupta, Madhu Sudan Kumar, and Prasanta K Jana. 2016. Transfer time-aware workflow scheduling for multi-cloud environment. In *Computing, Communication and Automation (ICCCA), 2016 International Conference on*. IEEE, 732–737.
- Y. Kang, Z. Zheng, and M. R. Lyu. 2012. A Latency-Aware Co-deployment Mechanism for Cloud-Based Services. In *2012 IEEE Fifth International Conference on Cloud Computing*. 630–637.
- James Kempf, Jari Arkko, Neda Beheshti, and Kiran Yedavalli. 2011. Thoughts on reliability in the internet of things. In *Interconnecting smart objects with the Internet workshop*, Vol. 1. 1–4.
- Weiwei Lin, Chen Liang, James Z Wang, and Rajkumar Buyya. 2014. Bandwidth-aware divisible task scheduling for cloud computing. *Software: Practice and Experience* 44, 2 (2014), 163–174.
- Redowan Mahmud, Mahbuba Afrin, Md. Abdur Razzaque, Mohammad Mehedi Hassan, Abdulhameed Alalaiwi, and Majed Alrubaian. 2016. Maximizing quality of experience through context-aware mobile application scheduling in cloudlet infrastructure. *Software: Practice and Experience* 46, 11 (2016), 1525–1545. spe.2392.
- Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. 2018. Fog computing: A taxonomy, survey and future directions. In *Internet of Everything*. Springer, 103–130.
- Takayuki Nishio, Ryoichi Shinkuma, Tatsuro Takahashi, and Narayan B. Mandayam. 2013. Service-oriented Heterogeneous Resource Sharing for Optimizing Service Latency in Mobile Cloud. In *Proceedings of the First International Workshop on Mobile Cloud Computing & Networking (MobileCloud '13)*. ACM, New York, NY, USA, 19–26.
- Architecture Working Group OpenFog Consortium. 2017. OpenFog Reference Architecture for Fog Computing. *OPFRA001 20817* (2017), 162.
- Beate Ottenwalder, Boris Koldehofe, Kurt Rothermel, and Umakishore Ramachandran. 2013. MigCEP: Operator Migration for Mobility Driven Distributed Complex Event Processing. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems (DEBS '13)*. ACM, New York, NY, USA, 183–194.
- Mrutyunjaya Sahani, Chiranjiv Nanda, Abhijeet Kumar Sahu, and Biswajeet Pattnaik. 2015. Web-based online embedded door access control and home security system based on face recognition. In *International Conference on Circuit, Power and Computing Technologies (ICCPCT)*. IEEE, 1–6.
- M. Slabicki and K. Grochla. 2016. Performance evaluation of CoAP, SNMP and NETCONF protocols in fog computing architecture. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 1315–1319.
- I. Takouna, R. Rojas-Cessa, K. Sachs, and C. Meinel. 2013. Communication-Aware and Energy-Efficient Scheduling for Parallel Applications in Virtualized Data Centers. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. 251–255.
- Mohit Taneja and Alan Davy. 2017. Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm. In *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 1222–1228.
- Ovidiu Vermesan and Peter Friess. 2014. *Internet of things-from research and innovation to market deployment*. River Publishers Aalborg.
- Shiqiang Wang, Rahul Uргаonkar, Ting He, Kevin Chan, Murtaza Zafer, and Kin K Leung. 2017. Dynamic service placement for mobile micro-clouds with predicted future costs. *IEEE Transactions on Parallel and Distributed Systems* 28, 4 (2017), 1002–1016.
- Cong Xu, Sahan Gamage, Pawan N. Rao, Ardalan Kangarlou, Ramana Rao Kompella, and Dongyan Xu. 2012. vSlicer: Latency-aware Virtual Machine Scheduling via Differentiated-frequency CPU Slicing. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing (HPDC '12)*. ACM, New York, NY, USA, 3–14.
- Sami Yangui, Pradeep Ravindran, Ons Bibani, Roch H Glitho, Nejib Ben Hadj-Alouane, Monique J Morrow, and Paul A Polakos. 2016. A platform as-a-service for hybrid cloud/fog environments. In *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 1–7.

Received February 2007; revised March 2009; accepted June 2009