Latency-Aware Application Module Management for Fog Computing Environments

REDOWAN MAHMUD, KOTAGIRI RAMAMOHANARAO, and RAJKUMAR BUYYA,
University of Melbourne

The 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 the fog computing environment, commonly named 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 interdependent Application Modules can be deployed in an orderly way over the nodes based on their latency sensitivity. In this article, we propose a latency-aware Application Module management policy for the fog environment that meets the diverse service delivery latency and amount of data signals to be processed in per unit of time for different applications. The policy aims to ensure applications' Quality of Service (QoS) in satisfying service delivery deadlines and to optimize resource usage in the fog environment. We model and evaluate our proposed policy in an iFogSim-simulated fog environment. Results of the simulation studies demonstrate significant improvement in performance over alternative latency-aware strategies.

CCS Concepts: • **Computer systems organization** → **Distributed architectures**; • **General and reference** → *Performance*; • **Computing methodologies** → Simulation evaluation;

Additional Key Words and Phrases: Internet of things, fog computing, application management, latency awareness, application placement, resource optimization, application QoS

ACM Reference format:
Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2018. Latency-Aware Application Module Management for Fog Computing Environments. *ACM Trans. Internet Technol.* 19, 1, Article 9 (November 2018), 21 pages.
https://doi.org/10.1145/3186592

1 INTRODUCTION

Due to rapid advancements in communication and hardware technology, it is expected that by 2020 there will be over 50 billion Internet of Things (IoT) devices with many real-time latency-sensitive applications (Vermesan and Friess 2014). In this context, clouds can be used as infrastructure for hosting IoT applications. However, as cloud data centers are geographically centralized in nature, it is difficult for them to support applications dealing with a large number of highly distributed IoT devices. In the long run, this inconvenience causes unacceptable high latency in service

Authors' addresses: R. Mahmud, K. Ramamohanarao, and R. Buyya, University of Melbourne, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information System, Parkville Campus, Melbourne, Victoria, 3010, Australia; emails: mahmudm@student.unimelb.edu.au, (kotagiri, rbuyya}@unimelb.edu.au.

1533-5399/2018/11-ART9 $15.00
https://doi.org/10.1145/3186592

delivery and congestion in networks. Therefore, to overcome these obstacles, fog computing was introduced (Bonomi et al. 2012).

Fog computing refers to a hierarchically distributed computing paradigm that bridges cloud data centers and IoT devices. The fog computing environment offers both infrastructure and a platform to run diversified software services. Hence, fog computing extends cloud-based facilities to the edge network, reduces service delivery latency and network congestion, and improves Quality of Service(QoS) (Dastjerdi et al. 2016). At different hierarchical levels of the fog environment, the physical devices are commonly called fog nodes. Traditional networking devices like set-top boxes, gateway routers, smart switches, and proxy servers, equipped with computational resources, can be used as potential fog nodes. Fog nodes are generally heterogenous in terms of resource capacity and application execution environment. Unlike cloud data centers, most of the fog nodes are heavily resource constrained due to 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 such fog computing environments, the applications are modeled as a collection of lightweight, interdependent 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, preprocessing and analysis of the received data, and handling events of interest (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 process input within a fixed time frame, each module requires a certain amount of resources, e.g., CPU, memory, bandwidth, and so forth. Hence, Application Modules together with their allocated resources constitute the data processing elements for different applications. This sort of decomposition is effective for distributed development of large-scale applications. In the literature, a similar concept is used to divide component-based applications into multiple Application Components (Yangui et al. 2016). However, while executing applications in a distributed manner, latency-related issues such as node-to-node communication delay, application service delivery deadlines, and service access frequency often become predominant and influence Quality of Service (QoS) and resource utilization. In different computing paradigms, although various latency-aware management strategies for distributed applications are proposed (Kang et al. 2012; Nishio et al. 2013; Ottenwälder et al. 2013; Takouna et al. 2013), the aforementioned latency issues have not been addressed simultaneously. Besides, due to dependency toward centralized management and lack of latency-sensitive application prioritization, the existing policies often get interrupted in meeting the challenges of IoT-enabled real-time interactions (Afrin et al. 2015). Therefore, in this article, we propose a latency-aware Application Module management policy for the fog computing environment that considers different latency aspects of distributed applications in a body with decentralized coordination. 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 while optimizing resources in fog computing. The main **contributions** of this article are:

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

The rest of the article is organized as follows. In Section 2, we highlight several relevant research works. In Section 3, different event-driven IoT application scenarios are 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 article.

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 the latency incurred due to service forwarding can be avoided.

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

Takouna et al. (2013) indicate toward communication latency and energy-aware placement of parallel applications in virtualized data centers. The proposed policy dynamically identifies the bandwidth demand and communication characteristics of the distributed applications and reallocates 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 a migration target by checking the resource availability.

Gupta et al. (2016) propose a transfer-time-aware workflow scheduling policy for the multicloud environment. It prioritizes interdependent tasks for scheduling to the multiclouds based on the computation cost and communication time. Within the clouds, the tasks are mapped considering the earliest start and finish time. The policy aims to enhance service delivery time. The relevant operations of the proposed policy are handled by a global cloud manager.

Fan et al. (2018) discuss data placement in geo-distributed multiclouds. The proposed Energy-Efficient Latency-Aware Data Deployment Algorithm (ELDD) sorts data chunks in nonascending order according to the collective access probability from all the users and merges them into larger data segments based on the capacity of the servers. Later it iteratively searches for appropriate servers where the placement cost of such data segments gets reduced. The placement cost is calculated centrally considering service access latency and energy consumption of the servers and the network. The algorithm also iteratively turns off 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 a hybrid cloud-fog environment. Being a centralized coordinator, the PaaS architecture allows developing applications according to the target domain; discovering, initiating, configuring, and scaling resources for deploying and executing the application components; managing execution flow between the components; monitoring SLA and component

migration; and 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 a cloud-fog environment. The work aims to ensure proper resource utilization. The algorithm is aware of the network issues. It sorts both the nodes and application modules according to the available capacity and requirements and maps the modules when the constraint is satisfied. In one sense, it prioritizes the modules based on the resource expectation. The policy reflects the way to reduce resource underutilization for distributed IoT applications. It also highlights how fog-cloud interoperation can minimize end-to-end latency compared to the cloud-based approaches.

Ottenwälder 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 a time-graph model to identify possible migration targets. Considering the shortest path from the data source, it selects the appropriate target instance from the time-graph model. On the selected instance, the policy applies coordination 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 proposed system, a coordinator manages all incoming requests and resources in order to meet service latency for different applications. While sharing resources, smaller amounts of tasks are forwarded from one node to another node under supervision of the coordinator. Policies running in the system trade off utility gain and energy consumption for resource optimization.

A cost-optimized offline and online service placement policy for the 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 with respect to 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 facilitate latency-aware scheduling of applications in virtual machines, Xu et al. (2012) introduce a scheduler named vSlicer. vSlicer nurtures the concept of *differentiated-frequency microslicing*. Unlike traditional schedulers, vSlicer divides a CPU slice into many microslices, and according to microslices, it schedules applications in higher frequency. By doing so, it increases the CPU access probability of applications.

Table 1 provides a brief summary of the state of the art for latency-aware application or service placement in distributed server, multicloud, mobile-cloud, and cloud-fog environments. Compared to the existing works, the unique aspect of our work is that we have considered service access delay, service delivery time, and internodal communication delay simultaneously while placing and forwarding interdependent application modules over distributed fog nodes. Besides, the proposed policy decentrally coordinates 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, and so forth. Our policy can place the modules both horizontally and vertically and in order to facilitate low energy usage, and can optimize resources by forwarding all the modules from one node to the others. Moreover,

Table 1. 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)	✓	✓	✓	✓	✓	✓	✓	✓

it enhances the priority of latency-sensitive applications to place them 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, module placement, and forwarding time.

3 APPLICATION SCENARIOS

3.1 IoT-Enabled Systems

In advanced healthcare and smart-home-based systems, the structure of different IoT applications is reflected 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 levels of asthma patients, pulse oximeters are widely used at hospitals. Usually pulse oximeters are connected to the bedside monitors and continuously show oxygen levels carried in the body, heart beat rate, and changes in blood volume of the skin (Addison et al. 2015). The bedside monitors provide the interface for authentication, aggregate data signals, and usually forward the sensed data to the 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 with respect to hypoxemia, hypercapnia, and so forth operate on the filtered data. Sometimes the analyzed outcome can indicate an emergency situation. Based on the outcome, required actions, for example, ventilation, injection, medication, and so forth, are triggered at the patient's bedside actuators. For critical asthma patients, a 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 applications 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 a smart-home-based system, usually entrance-side cameras take the pictures of the visitors and send them to the cloud or other computing entities for image processing (Sahani et al. 2015). Sometimes due to weather conditions and other external effects, the taken pictures get a lots of noise. In this context, image filtration techniques are required to apply for selecting the most appropriate picture and reducing its noise.

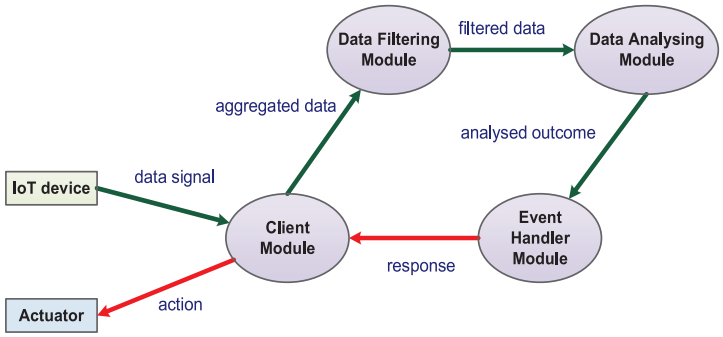

Fig. 1. Application model.

Image analytics with respect to face and gesture recognition, object detection, and so forth are also applied to the filtered pictures for identifying the visitor and the handheld objects. Once the visitor is identified and the handheld 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, a notification to the residents is created containing the details of the visitor. During the urgent period, a corresponding application of the Visitor Identification System is required to coordinate the aforementioned operations within a reasonable time that may not be possible if the application is placed in a 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 a *Client Module* (provides initial application interface), *Data Filtering Module* (applies data filtering techniques), *Data Analyzing Module* (executes data analytics), and *Event Handler Module* (generates appropriate response to the event). Data dependency exists among the modules of the same application, which can be expressed through a sequential unidirectional data flow as shown in Figure 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.

The 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 modules, and their interdependency are notified to the system through the Client Module. After deployment of all modules, the Client Module of a particular application directly communicates with respective IoT devices to grab the data signals and forwards in the 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, the Event Handler Module sends a corresponding response toward the Client Module. This response is eventually considered as the final application service for an IoT data signal. Based on the response, the Client Module triggers action in the actuators.

Since the 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 proximity of the users.

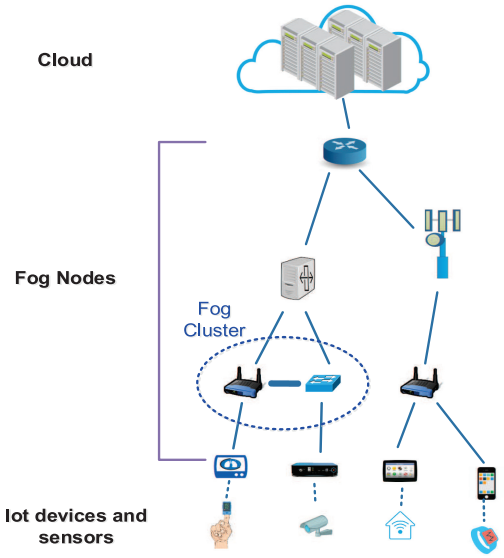

Fig. 2. Organization of fog.

4 SYSTEM MODEL AND ASSUMPTIONS

4.1 Organization of Fog Layer

In this work, the cloud is considered as a standalone 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 the cloud and IoT devices. Within this layer, nodes are organized in a hierarchical order as shown in Figure 2.

Lower-level fog nodes are closer to the IoT devices. As the level number goes higher, the 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 are 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 the 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), and so forth. (Slabicki and Grochla 2016). Therefore, maximum nodal communication delay ϵ_C within a fog cluster C is negligible and does not impact service delivery time extensively. We assume that at lower fog levels, if two nodes from the same level are connected with an identical uplink node and experience an 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 internodal latency (Kempf et al. 2011).

4.2 Fog Node Architecture

In order to define the architecture of fog nodes for distributed application management, we have extended the concept of the 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 Figure 3.

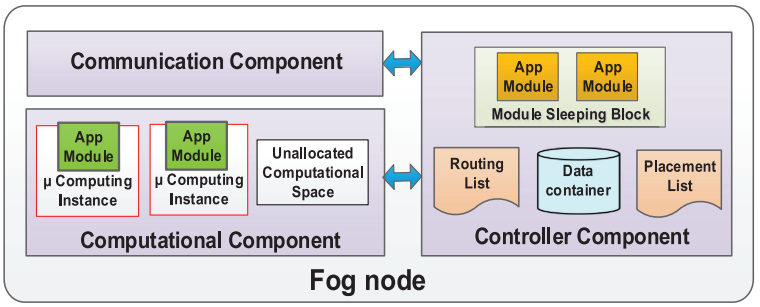

Fig. 3. Fog node components.

The Computational Component provides resources to execute Application Modules. Inside the Computational Component, modules are assigned to *Micro Computing Instances (MCIs)* where resources, e.g., CPU, memory, bandwidth, and so forth, are allocated according to the requirements. Due to resource constraints, each fog node can configure a certain number of individually working MCIs at a time. In a fog node when no MCIs are running, its Computational Component is turned off. In this case, the node only serves networking functionalities like routing, packet forwarding, and so forth through its Communication Component. If the load of applications increases in the fog layer, the Computational Component of that node can be turned on again to handle the event. The Controller Component of a fog node monitors and manages the operations of the Computational and Communication Components.

Moreover, the Controller Component maintains several data structures. Among them, the *Module Sleeping Block (MSB)* contains nonexecuting 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 the host (both node and MCI) of the next module, either the PL or RL of the corresponding fog node is referred to. Besides, a fog node preserves metadata of the placed modules even when they are no longer associated with it and the context information of other nodes in a *Data Container*.

The relevant notations and definitions used in modeling the system are listed in Table 2.

4.3 Latency Model

Due to data dependency, the generated output of an Application Module is sent to another module as input. The tolerable intermodule 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, $\delta_a^{ClientFilter} + \delta_a^{FilterAnalysis} + \delta_a^{AnalysisEvent} < \delta_a^{EventClient}$ is assumed so that tolerable intermodule data dependency delay-aware placement of the Data Filtering, Analyzing, and Event Handler Module can spontaneously justify placement of the Client Module.

However, an intermodule data dependency delay of 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 internodal communication delay $\Delta_{n'n}$ between their host nodes. In some cases, an input scheduling delay of the previous module can also contribute to the estimated intermodule data dependency delay of the respective module.

Table 2. 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 that belong 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 intermodule 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 intermodule data dependency delay of module m from its previous module m' on the data flow of application a; $m', m \in M_a$
$\Delta_{n'n}$	Internodal communication delay between node n' and n; $n, n' \in N$
f^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; $n', n \in N$
ϕ_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 $m \in M$
ψ_{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
ϵ_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
μ_n^m	Assigned MCI to module $m \in M$ in node $n \in N$
$y_n \in \{0, 1\}$	Equals 1 if node $n \in N$ is computationally active, 0 otherwise
$x_{mn} \in \{0, 1\}$	Equals 1 if module $m \in M$ is mapped to node $n \in N$, 0 otherwise
$x'_{mn} \in \{0, 1\}$	Equals 1 if module $m \in M$ was earlier deployed in node $n \in N$, 0 otherwise

In addition, the service access rate of data signals for an application and replication number of the previous module plays an important role in 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 remain idle for the next 0.3ms. Within this idle period of module m, its assigned MCI μ_n^m can be allocated to other modules 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 facilitates faster service access and delivery. Besides, not all applications show identical response to latency-related issues. For latency-sensitive applications, the service delivery deadline to the module's tolerable data dependency delay is stringent compared to latency-tolerant applications. In this case, placement of latency-tolerant applications in a limited number of lower-level fog nodes can obstruct many latency-sensitive applications to meet their requirements. Conversely, by considering the lower fog level scalable, if all applications are placed there, upper-level nodes will remain underutilized. Therefore, an efficient module placement policy is required that can prioritize applications to place them in closer

proximity of the data source, meeting necessary latency-related issues. More precisely, the policy should identify which applications (modules) should be placed at the lower fog level and which are required to move toward the upper level.

Moreover, to minimize energy usage and expenses in the fog environment, the number of computationally active nodes can be optimized. 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 a distributed environment like fog, if the decisions regarding application management are taken decentrally, both application placement time and overhead from the centralized controller will be reduced. Thus, application management can be done without relying on a single entity, although it will be very difficult to coordinate the nodes.

5 PROPOSED APPLICATION MODULE MANAGEMENT POLICY

Our proposed latency-aware Application Module management policy runs on the Controller Component of each fog node 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 are located at the lower fog level. However, placement of the module next to the Client Module also initiates from the Application gateway node. In order to initiate placement of a module, the fog node executes the *PlaceAppModules* procedure given in Algorithm 1.

The PlaceAppModules procedure takes the to-be-placed Application Module, m; its previous module, m'; and observed network delay, ω, as arguments.

As shown in Algorithm 1, the PlaceAppModules procedure basically consists of four steps:

At first, the procedure inquires about the context of the current node (line 2). If the current node is cloud, the rest of the unassigned modules will be placed there; otherwise, the procedure inquires about the 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, ω (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 η is set to $true$ (line 17) and availability of the resources in the current node is updated (line 18).

ALGORITHM 1: Module placement algorithm

```
1:  procedure PlaceAppModules(m, m′, ω)
2:      p ← this node
3:      if p = Cloud then
4:          place rest modules in cloud
5:          return
6:      q ← p.uplinkNode
7:      z ← m′.hostNode
8:      if m ≠ null then
9:          if φ_z^{m′} + Δ_pq + ω < δ_a^{m′m} then
10:             ω ← ω + Δ_pq
11:             p.RL.add(m, q)
12:             q.PlaceAppModules(m, m′, ω)
13:         else
14:             η ← false
15:             if r_req^m < r_avail^p, ∀r ∈ R then
16:                 p.PL.add(m)
17:                 η ← true
18:                 p.update(r_avail^p)
19:             else
20:                 for u := C_p.activeNodes do
21:                     if r_req^m < r_avail^u, ∀r ∈ R then
22:                         p.RL.add(m, u)
23:                         u.PL.add(m)
24:                         η ← true
25:                         u.update(r_avail^u)
26:                         break
27:             if η = false then
28:                 select node υ ∈ C_p.inactiveNodes
29:                 p.RL.add(m, υ)
30:                 υ.PL.add(m)
31:                 υ.update(r_avail^υ)
32:             m′ ← m
33:             m ← m′.getNext
34:             p.DeployAppModules(m, m′, ϵ_{C_p})
35:     else
36:         return
```

(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 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 ω in this case is considered negligible and set equal to ϵ of the Cluster.

Algorithm 1 can be extended to handle the scenario when there exist no inactive nodes to host a 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. It 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 Equation (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}. Binary variables x'_{mn} and $x'_{mn'}$ tracks whether module m had been available in node n or n' since the last placement. The constraints ensure that a module will not be mapped to multiple nodes Equation (2), resources of the target node satisfy the module's requirements Equation (3), placement of the module to target node does not affect the tolerable inter-module data dependency delay of the next module Equation (4) and the required time to forward the module from source node to target node fits within the input arrival interval of that module Equation (5).

$$\min \sum_{n \in N} y_n \tag{1}$$

subject to,

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

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

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

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

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 up to 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 τ 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 τ 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).

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. 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

ALGORITHM 2: Module forwarding algorithm

```
1:  procedure ForwardAppModules(m, a, τ)
2:      p ← this node
3:      q ← p.RL.get(m)
4:      if q ≠ null then
5:          send τ to m on node q
6:          return
7:      m'' ← m.getNext
8:      host_n ← null
9:      host_m ← null
10:     for n' := C_p.highNodes do
11:         for m' := M_n' do
12:             n'.getInfo(m')
13:             if t^now > t^last_m' + φ^m'_n' then
14:                 if t^last_m' + 1/f_m' − t^now > φ^m_n' then
15:                     if r^m'_req ≥ r^m_req, ∀r ∈ R then
16:                         if x_mn' = 1&γ^mm''_a ≤ δ^mm''_a then
17:                             if λ_m' = false then
18:                                 host_n ← n'
19:                                 host_m ← m'
20:                                 λ_m' ← true
21:                                 n'.updateInfo(m')
22:                                 break
23:         if host_n && host_m ≠ null then
24:             break
25:     if host_n && host_m ≠ null then
26:         schedule m in μ_host_m on node host_n
27:         send τ to m on node host_n
```

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 approarches 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 (Ottenwälder 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 at 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 3.

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 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,

Table 3. 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

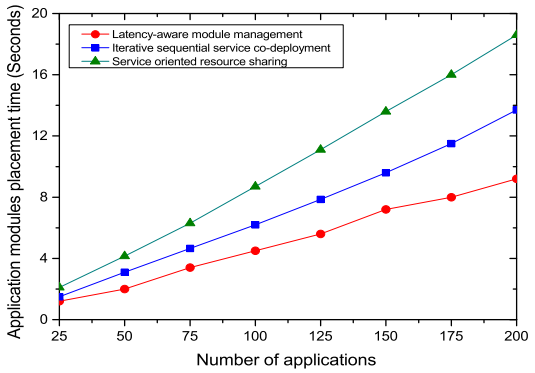

Fig. 4. Application Module deployment time vs number of applications.

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 time compared to others (Figure 4). This experiment also reflects that application placement decisions taken centrally can linger the placement of the applications in distributed Fog environment.

Figure 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 indicates that when in a system

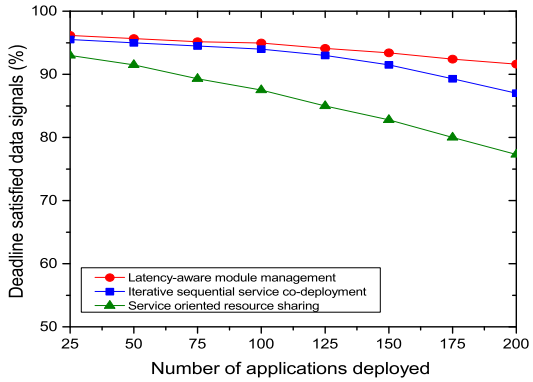

Fig. 5. Percentage of QoS satisfied data signals vs number of applications.

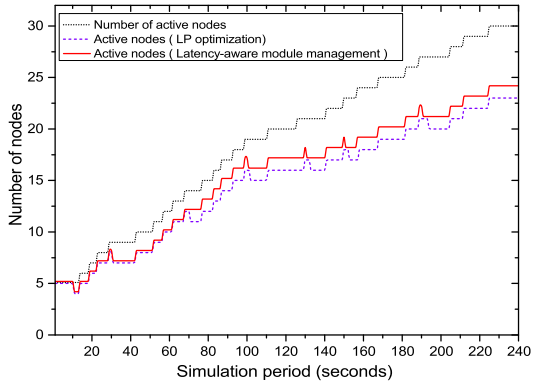

Fig. 6. Comparison of LP and heuristic based solution.

diversified applications in respect of latency-endurance exist, it is always a good policy to handle them separately.

6.3 Performance in Application Module Forwarding

Figure 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.

In Figure 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.

A comparative study of the proposed module forwarding approach, MigCEP and PVA is depicted in Figure 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

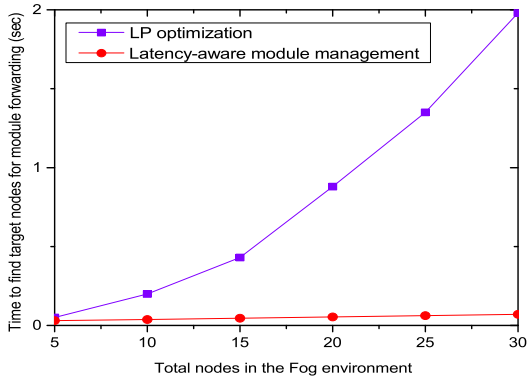

Fig. 7. Required time for generating LP and heuristic solution.

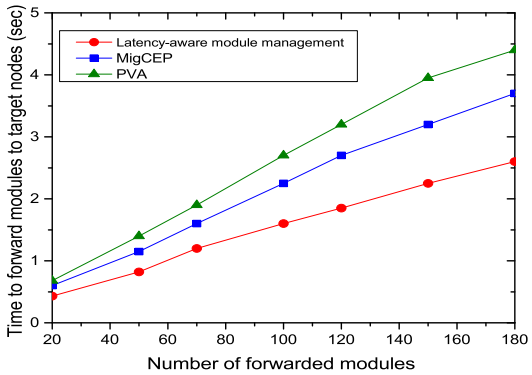

Fig. 8. Required time for Application Module forwarding vs number of forwarded modules.

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 clusters among Fog nodes can contribute extensively to forward modules so as to optimize resources.

Figure 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 proposed 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.

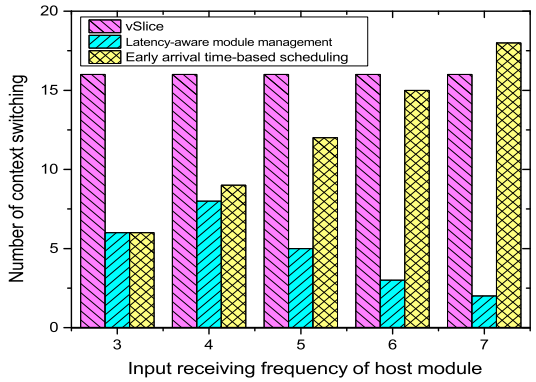

Fig. 9. Number of context switching vs frequency of host module.

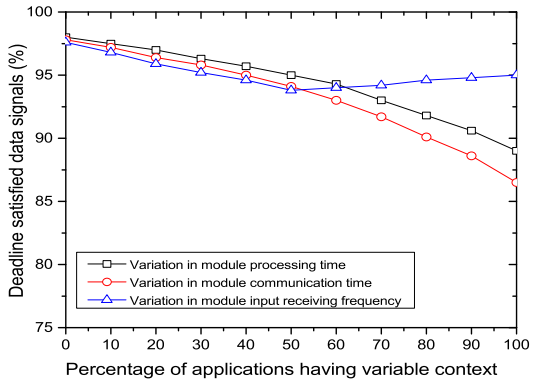

Fig. 10. Performance of the proposed policy for varying application context.

Figure 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, inter-nodal communication delay is considered as the dominating factor. Moreover, if the service access rate of the applications so as to the modules changes dynamically, initially QoS-degrades 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, Maria Salama and anonymous reviewers 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 (1 Feb 2015), 113–120.

M. Afrin, M. R. Mahmud, and M. A. Razzaque. 2015. Real time detection of speed breakers and warning system for on-road drivers. In *2015 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*. 495–498.

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, 43–55.

Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. ACM, 13–16.

Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. 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 G. S. S. Chalapathi. 2017. Latency aware mobile task assignment and load balancing for edge cloudlets. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 587–592.

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 *IFIP/IEEE International Symposium on Integrated Network Management (IM'09)*. 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 *International Conference on Computing, Communication and Automation (ICCCA)*. 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 Alelaiwi, 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.

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 Ottenwälder, 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 Urgaonkar, 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 March 2017; revised November 2017; accepted February 2018