Microservices-based IoT Application Placement within Heterogeneous and Resource Constrained Fog Computing Environments

Samodha Pallewatta,Vassilis Kostakos and Rajkumar Buyya
Cloud Computing and Distributed Systems (CLOUDS) Laboratory
School of Computing and Information Systems
The University of Melbourne, Australia
Email: ppallewatta@student.unimelb.edu.au, {*vassilis.kostakos,rbuyya*}*@unimelb.edu.au*

***Abstract*—Fog computing paradigm has created innovation opportunities within Internet of Things (IoT) domain by extending cloud services to the edge of the network. Due to the distributed, heterogeneous and resource constrained nature of the Fog computing nodes, Fog applications need to be developed as a collection of interdependent, lightweight modules. Since this concept aligns with the goals of microservices architecture, efficient placement of microservices-based IoT applications within Fog environments has the potential to fully leverage capabilities of Fog devices. In this paper, we propose a decentralized microservices-based IoT application placement policy for heterogeneous and resource constrained Fog environments. The proposed policy utilizes the independently deployable and scalable nature of microservices to place them as close as possible to the data source to minimize latency and network usage. Moreover, it aims to handle service discovery and load balancing related challenges of the microservices architecture. We implement and evaluate our policy using iFogSim simulated Fog environment. Results of the simulations show around 85% improvement in latency and network usage for the proposed microservice placement policy when compared with Cloud-only placement approach and around 40% improvement over an alternative Fog application placement method known as Edge-ward placement policy. Moreover, the decentralized placement approach proposed in this paper demonstrates significant reduction in microservice placement delay over centralized placement.**

***Keywords*-fog computing; internet of things (IoT); application placement; microservices architecture; application deployment**

I. Introduction

The emerging Internet of Things (IoT) paradigm enables creation of smart environments by interweaving sensors, actuators and data analytics platforms. CISCO forecasts the number of IoT devices and connections to rise to 14.6 billion by 2022, which would constitute half of the expected number of devices globally, and more than 6 percent of total ip traffic [1]. Sending such data from geo-distributed IoT devices towards the centralized Cloud would be inefficient due to network congestion and high latency. Since majority of the IoT applications are latency-sensitive and bandwidth-intensive, computation support closer to data source is vital to meet IoT application requirements (QoS, cost, etc.). To offer this facility, the computing paradigm called Fog computing is introduced [2].

Fog computing extends cloud-based services to the edge of the network by using devices that reside between IoT devices and Cloud [3]. Any device that has computational, networking and storage capabilities and lies in the path between IoT devices and Cloud can be considered as a Fog node. Unlike Cloud data centers, these devices are distributed, heterogeneous and resource constrained. Hence, IoT applications need to be modeled as collections of interdependent, lightweight modules that can be easily deployed onto these Fog nodes.

With the rapid evolution of IoT, developing applications as monoliths lead to poor scalability, extensibility and maintainability [4]. Thus, microservices approach has become increasingly popular in the development of cloud-centric IoT applications. According to M. Fowler and J. Lewis, microservice architectural style is defined as, "an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API" [5]. IoT applications in Fog environments can also benefit from this approach due to the following characteristics of microservices:

- Independently deployable - Microservices are easily containerizable by design due to them being loosely coupled, independent and self-sustained instances. In turn containers are suitable for Fog computing environments due to lower startup time, lower virtualization overhead and support for scalability [4].
- Independently scalable - Microservices support both vertical and horizontal scalability. Vertical scalability represents change of resource allocation as per the load whereas horizontal scalability indicates having multiple replicas of a single microservice to support the load. As Fog environments consist of resource constrained nodes that are heterogeneous in resource availability, horizontal scalability of microservices can have a great impact on increasing performance of applications deployed within Fog environments.
- Lack of centralized management - This matches with highly distributed nature of the Fog computing environments.

Thus, applications developed using microservices have the potential to be efficiently adapted to Fog environments. But, the introduction of microservices-based applications creates challenges in terms of service discovery, load balancing and decentralized management.

In literature, there are notable number of works that focus on developing placement algorithms for distributed applications within Fog-Cloud environments [6], [7], [8]. However, the placement of microservices-based Fog applications has not been investigated extensively. Existing works lack a decentralized approach for placing microservices within heterogeneous and resource-constrained Fog environments, focusing on horizontal scalability and challenges such as service discovery and load balancing. In our work, we present a microservices-based IoT application placement policy addressing above mentioned aspects.

Thus, **key contributions** of our work can be summarized as follows:

1) A decentralized placement algorithm for microservices-based IoT applications, highlighting horizontal scalability of microservices within resource constrained and heterogeneous Fog nodes.
2) A Fog node architecture to support decentralized placement along with service discovery and load balancing.
3) An implementation of our proposed policy on the iFogSim simulation environment and comparison against different placement approaches in terms of latency, network usage and efficiency of decentralization.

The rest of the paper is organized as follows. In section 2, we highlight related research. Section 3 presents microservices-based application model, Fog architecture and Fog node architecture along with the problem description. Our proposed solution is provided in section 4 along with relevant algorithms. In section 5 we present steps related to the implementation of the solution using iFogSim simulator whereas section 6 reflects simulation setup and performance evaluation. Finally, section 7 concludes the paper with future work and research directions.

II. Related Work

Microservices architecture is a recent concept that has created a phenomenal impact on development of IoT applications within Cloud computing environments. Butzin et al. [9] investigate the state of the art of IoT and microservices architecture to show how their architectural goals are quite similar. Several research studies have been conducted on developing Cloud-centric IoT frameworks based on microservices architecture [10], [11], [12], [13]. However, research on adapting this concept to Fog environments is still in its early stages.

Filip et al. [14] present a centralized placement approach for Edge-Cloud scheduling of microservices using Bag of Tasks (BoT) model where each task consists of one or more microservices. In the proposed architecture, nano data centers are used as Edge resources. Scheduling engine receives jobs and assigns them to VMs in the nano data centers or Cloud. Their scheduling policy places all microservices of a certain job within the same processing element or move it towards the Cloud, based on resource availability.

Santoro et al. [15] implement a framework and a software platform for orchestration of microservices-based IoT application workloads. In the proposed architecture, applications are modeled as a collection of microservices distributed via container images. The proposed architecture consists of IoT device layer, Edge gateways, Edge cloudlets and Cloud. Microservice deployment requests are sent towards a negotiator that accepts or rejects the requests. Orchestrator calculates the most suitable device to deploy microservices of the accepted requests, based on requirements (CPU, RAM, bandwidth, etc.) defined in deployment requests.

A centralized throughput aware placement algorithm for microservices-based Fog applications is presented in [16]. The proposed system consists of Edge servers that are grouped together based on their geographical regions. In this work, application microservices are placed within the region that contains respective IoT device or within the neighboring region. A greedy algorithm is presented for mapping these microservices onto Edge servers with sufficient computational resources while ensuring that bandwidth of the involved links can satisfy the throughput requirements of the application microservices.

Moreover, there are numerous works in literature that try to solve application placement problem in Fog environments by depicting applications in a distributed manner as a collection of interdependent modules. But, the concept of microservices architecture along with its unique characteristics and challenges are not observed in these works.

Taneja et al. [6] present a resource aware module mapping algorithm for placement of distributed applications within Fog environments. This work tries to optimize resource utilization by sorting application modules and nodes based on required resources and available capacity respectively and mapping sorted modules to resources. The proposed algorithm is compared with Cloud-only placement to depict the reduction of end-to-end latency in the Fog placement approach. This work defines a hierarchical Fog architecture where each Fog node is connected with a node in immediate upper layer of the hierarchy. Horizontal connections among Fog nodes of the same level are not defined. Moreover, placement is managed through a centralized approach.

Gupta et al. [7] propose a centralized edge-ward module placement algorithm for placing distributed applications modeled as Directed Acyclic Graphs (DAG). Their algorithm commences the placement of application modules starting from lower level Fog nodes and move upwards the hierarchy until a node with enough resources is met. But, their proposed algorithm supports only the vertical scaling of modules and does not consider horizontal connections among Fog nodes within the same Fog level.

Latency aware placement of application modules within Fog environments is presented in R. Mahmud et al [8]. This work, proposes a decentralized module placement method that considers service access delay, service delivery time and internodal communication delay when placing application modules within Fog environments. In this approach, distributed applications consisting of interdependent modules are placed and forwarded vertically and horizontally to satisfy the latency requirements of the application while optimizing resource usage.

A summary of the reviewed related works is presented in Table I, comparing them in terms of architecture of the Fog layer, application model and application placement approach. Architecture of the Fog layer used in each work is identified as hierarchical, if the Fog tier consists of multiple Fog levels

Table. I: Summary of Literature Study

Work	Fog Layer Architecture		Application Model		Decentralized Placement	Microservices-based Application
	Hierarchical	Clustering	DAG	BoT		
Taneja et al.(2017)	✓		✓			
Gupta et al. (2017)	✓		✓			
R. Mahmud et al. (2018)	✓	✓	✓		✓	
Filip et al. (2018)				✓		✓
Faticanti et al. (2018)			✓			✓
Santoro et al (2017)				✓		✓
Microservice Placement (this work)	✓	✓	✓		✓	✓

where each device is connected with a node in immediate upper layer and the latency from the IoT devices and resource availability of the nodes increase when moving towards upper levels. Clustering denotes existence of horizontal connections among Fog nodes of the same hierarchical level of the Fog architecture.

In our work, we present a placement algorithm for microservices-based IoT applications, where horizontal scalability of microservices is used within Fog node clusters that exist on the same hierarchical level of the Fog architecture. Moreover, the proposed policy also handles the challenges that come with horizontal scaling of microservices such as service discovery and load balancing. Our placement approach uses decentralized management of placement where each Fog node is responsible for placement decision making instead of having a centralized entity.

III. System Model and Problem Formulation

We propose a multi level, hierarchical Fog architecture where each Fog computing node is responsible for processing application placement requests. We model IoT applications as collections of containerized microservices and place them within the Fog environment using a decentralized placement approach. Our Fog architecture, application model, Fog node architecture and application placement problem are discussed in detail in the following subsections.

A. Fog Architecture

Fog computing makes use of computation, networking and storage capabilities of geographically distributed, heterogeneous and resource constrained devices such as mobile phones, access points, routers, proxy servers, nano data centers that span the continuum from IoT devices to Cloud. This, in turn provides localized services to end users, thus resulting in efficient bandwidth usage and low latency.

In this work, the three-tier hierarchical Fog architecture is used, where Fog layer is placed between IoT devices and Cloud data centers [17]. In our architecture, nodes within the Fog layer are also organized hierarchically as depicted in Fig. 1.

Fog nodes are placed in such a way that compute, storage and networking capabilities of Fog devices vary not only among the nodes in different levels but also within the same hierarchical level of the Fog layer. Compute, storage and network capability of Fog nodes increase when moving from lower levels to higher levels inside the Fog layer. Moreover, each Fog node has a direct connection with a node in immediate upper level and also

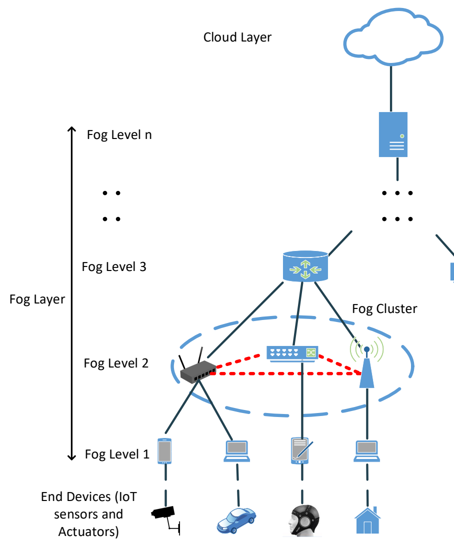

Fig. 1: Fog Architecture

can have links with nodes in the same level forming clusters among themselves. In this work, it is assumed that a certain Fog node belongs to only one cluster at a particular time. Nodes within the same Fog cluster communicates with each other using Constrained Application Protocol (CoAP) which is a simple web transfer protocol based on REST model [8], [18]. Therefore, the communication delay among cluster nodes is extremely low.

B. Application Model

In our work, we have modeled IoT applications as a set of microservices that can be deployed, upgraded and scaled independently. Each microservice is deployed as an independent container and the resource requirement for each microservice is defined in terms of CPU, bandwidth, RAM and storage. Fig. 2 depicts our microservices-based Fog application architecture. Each application consists of a Client module that is deployed onto end user devices such as mobiles, tablets etc. that reside in the lowest level of the Fog layer. This module is responsible for sending data received from IoT sensors, towards relevant microservices for processing and also displaying results or sending resulting signals to the actuators. The rest of the microservices are deployed on either Fog or Cloud layer based on the placement policy. Since microservices that make up

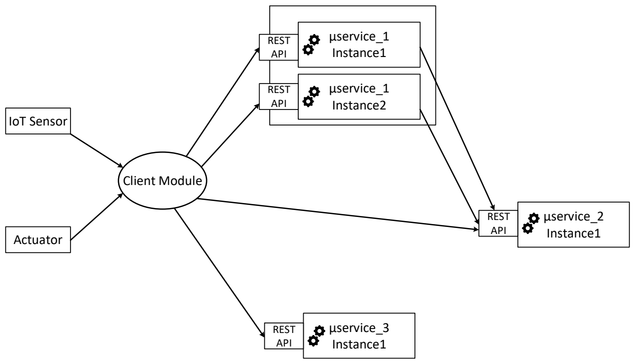

Fig. 2: Microservices-based IoT Application

an application have data dependencies amongst them, an IoT application is depicted as a Directed Acyclic Graph (DAG) [6]. In this model, each microservice is represented by vertices of the DAG, whereas edges between vertices represent data dependencies among microservices.

In microservices-based applications, microservices call each other through REST APIs to perform tasks. As microservices are horizontally scalable, a certain microservice can have multiple replicas to support the load balancing. When directing requests to microservices, two approaches are available: through server-side load balancing or client-side load balancing. In the server-side load balancing approach, a dedicated load balancer component lies between client microservices and server side microservices. This component is responsible for service discovery and directing the requests according to a load balancing policy. However, in a highly distributed environment such as Fog, using such centralized load balancing approach is not efficient. Thus, within this model, service discovery and load balancing is handled through a decentralized method by using client-side load balancing. So, in the proposed model, device that hosts the client microservice has to be aware of all microservice instances of the required services and route requests according to the load balancing logic.

C. Fog Nodes

In this paper, we define the Fog node architecture as an extension of [8] where each Fog node consists of three components: communication component, computational component and controller component. We extend this concept to support microservices architecture and propose an algorithm for microservices-based IoT application placement within Fog environments.

According to our model, *Placement of Microservices*, *Service Discovery* and *Load Balancing* are handled by controller component in the node. When a placement request is received by a Fog node, it is queued in *Placement Request Queue (PRQ)* in the controller component. Placement requests in the queue are processed one after the other using *Application Placement Logic*. *Service Discovery Info (SDI)* data block is a service registry that contains network locations of service instances that can be used by client microservices placed within the Fog node. Each time a client microservice makes a request, it is routed to a service instance according to the *Load Balancer Logic* using data in *SDI*.

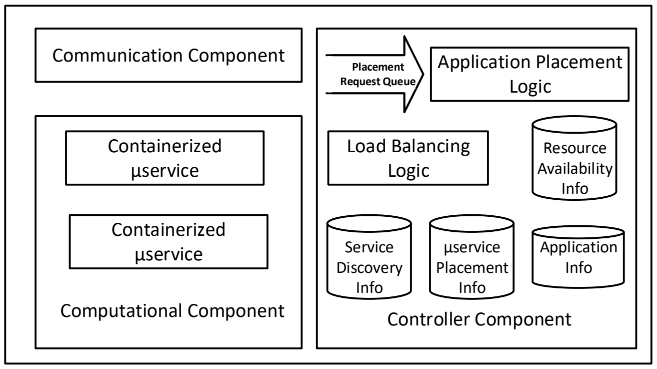

Fig. 3: Fog Node Architecture

Each node keeps track of available resources (*Resource Availability Info*) such as CPU, RAM, bandwidth and storage. This information is used when placing microservices and also when making decisions on scaling microservices across clusters of nodes that are in the same Fog level. *μservice Placement Info (μPI)* keeps track of all microservices that are placed within the Fog node along with resources allocated for each microservice. *Application Info* stores DAG representations of each IoT application available for placement within the Fog environment.

Computation component of the Fog node consists of deployed microservices. Each Fog node deploys microservices using container images that are available in a centralized container image registry.

D. Placement Problem

Placing latency critical and bandwidth hungry microservices belonging to IoT applications within lower levels of Fog layer results in reduction of latency and network usage. But Fog nodes that reside in the lower levels of the hierarchy are more resource constrained when compared with upper level Fog devices and Cloud data centers. Even within the same level, resource availability in Fog nodes varies. Since IoT end devices are highly distributed and dynamic, load on each of these Fog nodes also varies.

Under these circumstances, lower level Fog nodes may not be able to support the service demand which results in microservices being placed at higher levels in Fog hierarchy. Moreover, due to resource heterogeneity of nodes and varying loads on each Fog node, some nodes within same Fog level can have under-utilized resources while others fail to support the service demand. This can be overcome by creating clusters among Fog nodes of the same hierarchical level and scaling application modules among the clustered devices to support the load. This has several associated challenges noted below.

1) An efficient application microservice placement algorithm is needed that can identify what application microservices to be scaled and in which device in the cluster to place them.

Algorithm 1 Process Placement Request

```
Input: placement request $pr$
Output: placement request status; *Status.COMPLETED* for request processed, *Status.HALTED* for waiting for cluster placement
1: procedure PROCESSPLACEMENTREQUEST($pr$)
2:     $node \leftarrow this.node$
3:     $a \leftarrow pr.applicationId$
4:     $m_p \leftarrow pr.placedMicroservices$
5:     $m_f \leftarrow \{\}$                    ▷ Placement failed $\mu$services
6:     $m_{toPlace} \leftarrow Get\mu servicesToPlace(a, m_p, m_f)$
7:     while $m_{toPlace}$ is not $empty$ do
8:         if $node$ is $cloud$ then
9:             place all remaining microservices here
10:            send service discovery info
11:            return Status.COMPLETED
12:        else
13:            $m \leftarrow m_{toPlace}.remove(0)$
14:            $placementStatus = PlaceMicroservice(m)$
15:            if $placementStatus = Status.PLACED$ then
16:                $nodes_{client} = GetClientNodes(m, m_p)$
17:                for every node $n$ of $nodes_{client}$ do
18:                    $n.SDI.add(m, node)$
19:                $m_p.add(m, node)$
20:                if $m_{toPlace}$ is $empty$ then
21:                    $m_{toPlace} \leftarrow Get\mu servicesToPlace(a, m_p, m_f)$
22:            else if $placementStatus = Status.CLUSTER$ then
23:                return Status.HALTED
24:            else if $placementStatus = Status.FAILED$ then
25:                $m_f.add(m)$
26:                if $m_{toPlace}$ is $empty$ then
27:                    $m_{toPlace} \leftarrow Get\mu servicesToPlace(a, m_p, m_f)$
28:    if $m_p.size() < app\mu serviceCount(a)$ then
29:        $node_{parent} \leftarrow node.parent$
30:        $node_{parent}.PRQ.add(pr)$
31:    return Status.COMPLETED
```

2) A microservice discovery method to be used by client microservices to call server microservices.
3) A Load balancing mechanism to direct requests to scaled microservice instances.
4) A decentralized approach to meet above challenges.

In this paper, we propose a Microservice Placement Algorithm addressing aforementioned challenges.

IV. Proposed Solution

To solve the placement problem, we propose a heuristic placement algorithm, that scales microservices across Fog device clusters to accommodate load within resource constrained and heterogeneous Fog environments. The algorithm facilitates decentralized placement of microservices, service discovery and load balancing.

A. Microservice Placement

When a sensor joins a lower level Fog node, placement process is invoked by the corresponding Fog node. This Fog node which acts as the gateway to the rest of the Fog network, hosts the client module of the IoT application and rest of the module placement is carried out according to the *Application Placement Logic* starting from it. Gateway Fog node generates a *Placement Request (pr)* which consists of *Application id, Placed microservices map, Gateway device id* and *Placement request id*. *Application id* identifies each IoT application

Algorithm 2 Place Microservice

```
Input: microservice to place $m$
Output: microservice placement status : *Status.PLACED* if placed on this node, *Status.CLUSTER* if placement on cluster nodes and *Status.FAIL* if no resources are available on this node and cluster nodes
1: procedure PLACEMICROSERVICE($m$)
2:     if instance of $m$ already in node then
3:         if $req(m) \leq availCap(node)$ then
4:             increase resources allocated for instance of $m$
5:             $\mu PI.add(m)$
6:             return $Status.PLACED$
7:         else if node is in a cluster then
8:             send $ClusterPlacementQuery$ to cluster nodes
9:             return $Status.CLUSTER$
10:    else
11:        if $req(m) \leq availCap(node)$ then
12:            place $m$ on $node$
13:            $\mu PI.add(m)$
14:            return $Status.PLACED$
15:        else if node is in a cluster then
16:            send $ClusterPlacementQuery$ to cluster nodes
17:            return $Status.CLUSTER$
18:    return $Status.FAILED$
```

Algorithm 3 Place Microservice On Cluster

```
Input: microservice to place $m$
Input: cluster nodes $C$
1: procedure PLACEONCLUSTER($m, C$)
2:     $node \leftarrow this.node$
3:     $C' \leftarrow C.requestQueueEmptyNodes$
4:     for every node $n$ of $C'.nodesWithInstanceOfm$ do
5:         if $req(m) \leq availCap(n)$ then
6:             $n.PRQ.add(pr)$
7:             ProcessPlacementRequest(node.PRQ.dequeue())
8:             return
9:     for every node $n$ of $C'.activeNodesWithoutInstanceOfm$ do
10:        if $req(m) \leq availCap(n)$ then
11:            $n.PRQ.add(pr)$
12:            ProcessPlacementRequest(node.PRQ.dequeue())
13:            return
14:    for every node $n$ of $C'.inactiveNodes$ do
15:        if $req(m) \leq availCap(n)$ then
16:            $n.PRQ.add(pr)$
17:            ProcessPlacementRequest(node.PRQ.dequeue())
18:            return
19:    $node_{parent} \leftarrow node.parent$
20:    $node_{parent}.PRQ.add(pr)$
21:    ProcessPlacementRequest(node.PRQ.dequeue())
```

uniquely. Each Fog node has information on available IoT applications including microservices that form the applications and connections among those microservices in the form of a DAG which can be accessed using *Application id*. *Placed microservices map* consists of already placed microservices with respect to the placement request and nodes they are placed on. *Placement request id* is a unique id generated per request. It can be used to uniquely identify each sensor that joins the gateway node. Gateway Fog node sends this placement request towards the parent node where it gets added to parent node's *PRQ*.

PRQ is a First in First Out (FIFO) data structure. Thus, if the queue is not empty, requests are processed starting from the first request in the queue. Each request gets processed according to the Algorithm 1. For the selected placement request, algorithm determines the microservices to be placed based on the DAG representation of the application stored in *Application Info* (line 6). A microservice is selected for placement only if all the client microservices in the application that uses its service are already placed. $Get\mu servicesToPlace$ method traverse the DAG of the application and identifies such microservices, taking already placed microservices and placement failed microservices into consideration. Once the microservices are determined, placement begins from the current node. If current node is Cloud, all the remaining microservices are placed there (line 8-11), otherwise algorithm tries to place the selected microservices on the current node by calling *PlaceMicroservice* procedure (line 14) for each microservice in selected set of microservices, starting with the first in the set. If the placement succeeded, then the next microservice to place is found and placement process on the current node continues (line 15-21). If cluster placement is invoked (*Status.CLUSTER)*, then placement request processing is halted until cluster placement decision is made (line 22-23). If placement failed, then Algorithm 1 tries to place other possible microservices on the current node (line 24-27). After placing all possible microservices on the current node, pr is sent towards the parent node to place rest of the microservices of the application or pr processing is finished if all microservices of the application are placed (line 28-31). If Algorithm 1 returns $Status.COMPLETED$, next request in *PRQ* is selected for processing.

Microservices placement on each Fog device is carried out according to the Algorithm 2. If current node already contains an instance of the microservice, placement policy tries to scale the microservice. At this point microservice is either scaled vertically or horizontally. If considered node has requested amount of resources, allocated resources for the microservice are increased (line 3-5) whereas if not, microservice is scaled across the cluster to accommodate the load (line 7-8). If the node does not already contain an instance of the microservice, algorithm tries to place microservice on current node or on any of the nodes within Fog node cluster (line 11-17). If horizontal placement within a particular Fog level is not possible, procedure returns *Status.FAILED*, so that the pr is sent to the next level towards the parent node of the current Fog device.

When a Fog node does not have enough resources to support placement of a microservice, our proposed placement policy checks whether this node is in a cluster and if so tries to place the microservice within cluster nodes. To achieve this, a *Cluster Placement Query* is sent to all nodes in the cluster, to which cluster nodes reply with information on available resources (from *Resource Availability Info*), microservices already deployed on the node (from *µservice Placement Info*) and current *PRQ* size. Once replies from all the cluster nodes are received, Algorithm 3 is used to determine the suitable Fog node to place the microservice. For placement, cluster nodes with *PRQ* size of zero is considered. Here priority is given to nodes that already have required microservice placed on the device (line 4-8). If it failed, other active nodes in the cluster are considered (line 9-13). Inactive nodes are considered if this failed (line 14-18). Here inactive Fog nodes are the devices that does not have any microservices deployed and has no placement requests in *PRQ*. Once the cluster node selection is completed, current pr is sent either towards a cluster node or the parent node and the next placement request in the queue is taken for processing by the current node.

The proposed placement policy propagates pr until all microservices in the application are placed or scaled to support processing of the data generated by newly joined sensor. Once all microservices are placed, placement completion is informed to the gateway node along with *placement request id* of the request. Afterwards, gateway node starts accepting data from the associated sensor, identified based on the *placement request id*.

B. Service Discovery

With the proposed placement approach, as more IoT devices join the Fog environment, microservices get scaled across Fog devices. Thus, a client Microservice can send API requests to multiple service instances. As a result, service discovery is required for client microservices to identify available services. Handling this through a centralized method is not very efficient due to highly distributed nature of the Fog nodes. In our work we propose a decentralized approach as a solution to this challenge.

Every time a microservice is placed or scaled, *Placed microservices map* in the placement request can be used to find nodes that hosts the client microservices. Afterwards, each of these nodes are notified of the service placement. This information is stored within the *SDI* data structure of the recipient nodes.

In this approach, service discovery message is sent not only when a microservice is deployed, but also when resources allocated for a deployed microservice is increased to support a placement request. Thus, client can receive multiple service discovery messages with reference to a service deployed on a certain node. The number of such messages received by a Fog device acts as an indication of the amount of resources allocated for a service instance deployed on a certain node to handle the client requests. Hence, it is also stored within *SDI* and used later as the weighting factor for load balancing.

C. Load balancing

Information stored in SDI of each Fog node is used for load balancing. When making calls to services, this data is used, and API call is directed based on Load Balancing Logic. In this work, we've used a Weighted Round Robin method for load balancing where weighting is done based on the amount of resources allocated for each available service instance, which is stored within *SDI*.

D. Time Complexity Analysis

Time complexity of our microservice placement algorithm is analysed under two phases; time complexity of placement request processing which is handled by Algorithm 1 and Algorithm 2, time complexity of selecting a cluster node for the placement of a microservice covered by Algorithm 3.

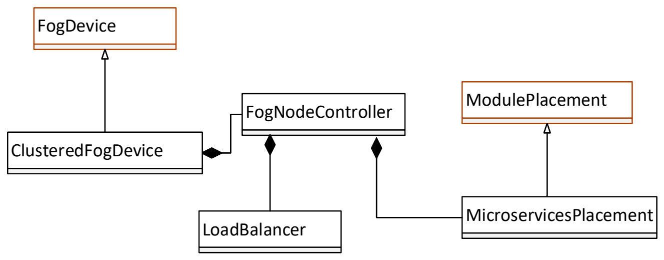

Fig. 4: Class diagram of extensions made to iFogSim Simulator (Existing classes: FogDevice.java and ModulePlacement.java)

In Algorithm 1, *GetμservicesToPlace* procedure removes placed microservices (m_p) from the application DAG and traverse the resultant DAG to find vertices without any incoming edges, while taking placement failed microservices of the current node (m_f) into consideration. If the application consists of *M* microservices that represent vertices of the DAG and *E* connections among them that represent edges of the DAG, above function has time complexity of *O*($|M| + |E|$). *PlaceMicroservice* function in Algorithm 2 is completed in constant time with complexity of *O(1)*. Hence, the time complexity of processing a placement request is *O*($|M| * (|M| + |E|)$).

For a Fog node cluster with C number of nodes, worst case time complexity of Algorithm 3 is of linear time. Thus, the time complexity of selecting a cluster node for the placement of a microservice is *O*($|C|$).

V. Design And Implementation

To evaluate the performance of the proposed policy, we implemented and simulated a Fog computing environment using iFogSim Simulator [7]. iFogSim is a simulation toolkit developed for the simulation of Fog environments. It is built based on CloudSim simulator [19] which is widely used for evaluating resource-management and scheduling policies for Cloud computing environments. iFogSim supports creation of hierarchical Fog architectures, modeling of distributed applications and evaluation of scheduling policies based on performance metrics such as latency, network usage and power consumption. Since these features are significant in modeling the proposed system, iFogSim was chosen for simulations. Moreover, several features were added to iFogSim simulator to support modeling of the proposed system. Fig. 4 represents new classes implemented within iFogSim simulator to support our placement policy.

iFogSim supports a centralized approach for application module placement where module placement is handled by a broker that has knowledge of overall Fog architecture and resource availability of each Fog node. Since our placement approach is decentralized, simulator was extended to support this. Instead of using the existing broker class (FogBroker.java), Fog nodes were implemented according to the Fog node architecture introduced in Fig. 3. Thus, in our implementation, each Fog node has a Fog node controller (FogNodeController.java) that handles microservices placement (MicroservicesPlacement.java) and load balancing (LoadBalancer.java).

iFogSim provides capabilities to create hierarchical Fog architectures with multiple Fog levels. But connections are made only vertically. Horizontal connections within the same Fog level are not available. Thus, clustering of Fog nodes within the same level cannot be simulated using current iFogSim version. So, the simulator was extended to support creation of clusters by forming connections among nodes of the same Fog level.

In iFogSim, data streams are realized using an object that is characterized by source and destination application modules. As a result, when a workload is simulated, data streams are always sent up the hierarchy till a Fog node that hosts the destination module is met. This implementation is not compatible with the proposed solution due to horizontal scaling and load balancing features introduced in our policy. This requires the simulator to direct data streams based on destination device instead of destination module. Moreover, due to clustering, data streams need to be routed to clustered nodes through horizontal links as well. These features were also added to the simulator to simulate the proposed placement policy.

In the proposed model, applications are developed as a collection of microservices where each microservice is deployed on a separate container using operating system level virtualization. In iFogSim, distributed applications are modeled as a collection of modules (AppModule.java), where resource requirement of each module can be defined. Even though AppModule class is implemented as an extension of VM class in CloudSim, it can be realized as OS level virtualization of containers by defining resource requirements accordingly and changing startup delay to match that of containers.

VI. Performance Evaluation

We evaluated our Microservice placement policy through simulation of a smart healthcare application and compared it with two existing application placement algorithms.

A. Experimental Configurations

To evaluate the performance of the proposed placement algorithm, we have used a synthetic workload generated by modeling a smart healthcare application on "*ECG Monitoring*" [20], [21]. Application is modeled according to the Microservices-based IoT application architecture mentioned earlier in Fig. 2. This application uses a wearable ECG sensor that transmits data towards Level 1 Fog nodes using Bluetooth technology. Application consists of two microservices, *ECG Feature Extraction Microservice* which extracts features from ECG to detect and notify about any existing abnormal situations and *ECG Analyser Microservice* which carries out further analysis on ECG data collected and stored for a longer duration of time. *ECG Feature Extraction Microservice* provides a latency critical service and is placed on either Fog layer or Cloud according to the placement policy. *ECG Analyser Microservice* receives results from *ECG Feature Extraction Microservice* where it further processes the extracted data, so that they can be

Table. II: Simulation Parameters

Parameter	**Value**
Latency values:	
IoT device to Fog Level 1	5ms
Fog Level 1 to Fog Level 2	20ms
Fog Level 2 to Fog Level 3	30ms
Fog Level 3 to Fog Level 4	50ms
Fog Level 4 to Cloud	150ms
Among cluster nodes	2ms
ECG sensor data transmission interval	5ms
Total number of ECG sensors connected	60
Container startup time	300ms
Placement Calculation time of a microservice	2ms
Simulation Time	120s

used by remote health monitoring purposes of hospitals. Thus, this microservice is placed on Cloud.

Fog environment modeled for simulations consists of four Fog layers with devices that are heterogeneous to each other in terms of resource availability. Clusters are formed between Fog devices in Fog Level 2 that are connected to the same Fog Level 3 device. Table II contains the parameters used in creating the simulated Fog environment.

B. *Results and Analysis*

Performance of the proposed placement policy is evaluated based on three performance metrics: latency of the latency critical path of the modeled application, network usage and required time for application microservice placement. To evaluate performance based on latency and network usage, the proposed Microservice placement policy is compared with two other placement approaches.

1) Cloud-only placement - All microservices of the application are placed within Cloud layer.
2) Edge-ward placement proposed in [7] - In this algorithm horizontal placement of the microservices across Fog node clusters is not considered. If a microservice placed on a certain Fog device does not have enough resources to handle the load, that microservice gets moved up the Fog hierarchy until a device that can handle the load is met.

The proposed Microservice placement policy targets to optimize placement within Fog environments that consist of heterogeneous and resource constrained Fog nodes. Thus, three scenarios that capture the above mentioned aspects, were used for the evaluation of the proposed placement policy.

1) Scenario 1 - Nodes on Fog level 2 have same resource capacities. But the number of Fog level 1 nodes per each Fog Level 2 node differs.
2) Scenario 2 - Nodes on Fog level 2 have same number of Fog Level 1 nodes connected. But resource capacities among Fog Level 2 devices differ.
3) Scenario 3 - Both resource capacity and number of connected Fog Level 1 nodes differ among Fog Level 2 nodes.

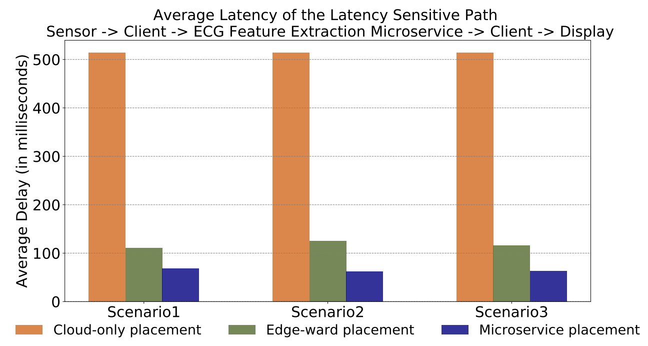

Fig. 5: Average Delay for Latency Sensitive Path

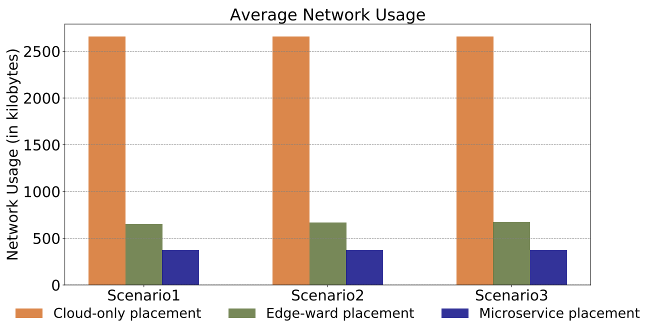

Fig. 6: Average Network Usage

All three scenarios depict heterogeneous and resource constrained Fog environments where some of the nodes get overloaded whereas others are under-utilized.

For these three scenarios, average latency of the latency sensitive loop (Fig. 5) and average network usage (Fig. 6) were measured after simulations.

In all three scenarios, Cloud-only placement shows a significant increase in both latency and network usage when compared to Edge-ward placement and Microservice placement approaches. Moreover, Microservice placement approach proposed in this paper outperforms both Cloud-only placement and Edge-ward placement approaches in terms of both latency and network efficiency.

In Cloud-only placement, as all microservices are placed within Cloud layer, data generated by geo-distributed sensors have to be sent towards centralized Cloud which is multiple hops away from the edge of the network. Since all the generated data are sent towards Cloud, amount of data flowing through the core network increases, resulting in network congestion. Due to these two reasons, latency of the services deployed on Cloud is significantly higher than other two scenarios where latency critical microservice is placed within Fog layer closer to the data source.

Raw data transmitted from IoT sensors requires a large amount of bandwidth. In the modeled application *ECG Feature Extraction* microservice, analyses raw ECG data and produce results. As a result, large volumes of sensor data get reduced into meaningful information and these information gets sent towards the *ECG Analyser Microservice* and the display. Thus, if the *ECG Feature Extraction Microservice* is placed at the Fog layer, volume of data transmitted through the core network reduces dramatically. This results in efficient utilization of bandwidth in Fog placement approaches when compared with Cloud-only placement.

In Edge-ward placement, horizontal scaling of microservices is not considered. So, if a certain instance of a microservice deployed on a fog node does not have enough resources available to handle received workload, that particular microservice is moved up the Fog hierarchy to a node with higher resource capacity. In contrast to this, the Microservice placement approach utilizes horizontal scaling and load balancing. If a certain microservice instance does not have enough resources to support the workload, microservice is horizontally scaled across nodes within clusters. These nodes are in the same Fog level and latency among nodes within the same cluster is extremely low due to the use of light weight web transfer protocols such as CoAP. Thus, our proposed approach utilizes microservices architecture to place latency critical services within lower Fog levels closer to the data source.

In all three scenarios, due to heterogeneity among resource constrained Fog nodes and difference of load on nodes, some Fog nodes gets over-utilized while others are under-utilized. Under such circumstances, proposed approach ensures that microservices are deployed in such a way so that available resources within Fog node clusters are utilized before moving towards higher level Fog nodes with higher resource availability. As a result, Microservice placement approach places modules in lower Fog levels when compared to Edge-ward placement. This results in further reduction of latency and network usage when using the proposed placement policy.

Based on the generated results, our proposed Microservice placement policy demonstrates around 85% improvement over Cloud-only placement and around 40% improvement over Edge-ward placement policy, in terms of both latency and network usage.

To evaluate the efficiency of using a decentralized placement approach, we evaluated the proposed method based on total time taken to place *ECG Feature Extraction* microservice within Fog layer. We implemented the same placement algorithm using a centralized placement method and compared the placement delay with the proposed decentralized approach.

In modeling the centralized placement method, a separate Fog node on Fog Level 4 was chosen as the centralized application scheduler. Once an ECG sensor joins the network, a placement request is sent by Fog Level 1 node towards this scheduler node. It maintains a complete view of the Fog hierarchy below Fog Level 4 and calculates the nodes to place the requested microservices. This decision is sent towards the selected nodes in order to deploy an instance of a microservice on respective Fog nodes.

Experiments were carried out changing the number of sensors connected to the Fog environment. Number of sensors were increased by increasing number of Fog Level 1 devices connected to each Fog Level 2 device where each Fog Level 1 device has one ECG sensor connected to it.

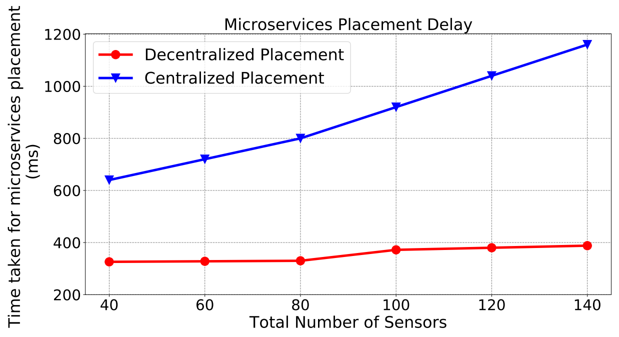

Fig. 7: Total time taken for deployment of Microservices within Fog layer

As per Fig. 7, placement delay of the centralized method is significantly higher than the decentralized placement. In centralized approach, all placement requests have to be sent towards Fog Level 4 scheduler node which is multiple hops away. After processing the request and selecting a Fog node to place the microservice, scheduler node has to inform this to the selected node placed within lower Fog levels. This induces a communication delay on all placement requests. But, in decentralized approach, placement process starts from Fog Level 1 node and propagates up the hierarchy until a suitable node is met. This results in significantly lower placement delay in decentralized placement.

Moreover, in centralized management all requests are sent towards a central scheduler node. Thus, as the number of sensors increases, the number of placement requests that needs to be processed by the centralized scheduler is higher than that of each Fog node in decentralized case. As a result, there's a rapid increase in placement delay for centralized management whereas increase of delay is quite small in decentralized approach as the number of sensors increases.

As the number of placement request increases, placement of microservices is moved towards upper level Fog nodes with higher resource availability. Slight increase of placement delay in decentralized placement is caused due to this. Our results show that due to highly distributed nature of the Fog nodes it is much efficient and scalable to use decentralized placement and service discovery methods.

VII. Conclusions and Future Work

Microservices-based IoT applications have the potential to improve the efficiency of IoT application placement within Fog environments. We propose a decentralized placement algorithm for microservices-based IoT applications, highlighting horizontal scalability of microservices within resource constrained and heterogeneous Fog nodes. Moreover, we also propose a Fog node architecture to support the proposed decentralized placement along with service discovery and load balancing. We conducted simulation-based experiments using iFogSim simulated Fog environment to demonstrate the performance of the proposed solution. Based on the results obtained through simulations, the proposed placement policy demonstrated significant reduction in latency and network usage within heterogeneous and resource constrained Fog environments. We also compared our approach with a centralized placement approach, which highlighted the suitability of decentralized management within Fog environments in terms of application placement delay and scalability of placement.

In future work, we plan to explore following research directions; implement the proposed policy in real world using FogBus [22], which is a lightweight framework developed for Fog computing; improve our microservice placement policy to adapt to dynamic characteristics such as Fog node failures and mobility of IoT devices and lower level Fog nodes; use dynamic clustering techniques to support efficient microservice placement; QoS aware prioritizing and placement of multiple application placement requests; energy efficient placement of microservices-based IoT applications within Fog environments and cost optimization of microservice placement within Fog environments.

Acknowledgments

We thank Redowan Mahmud and Mohammad Goudarzi for their valuable comments and suggestions on improving this paper.

References

[1] Cisco, "Cisco visual networking index: Forecast and trends, 2017–2022," *White Paper*, vol. 1, 2018.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "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, 2012, pp. 13–16.

[3] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of everything*. Springer, 2018, pp. 103–130.

[4] C. T. Joseph and K. Chandrasekaran, "Straddling the crevasse: A review of microservice software architecture foundations and recent advancements," *Software: Practice and Experience*, vol. 49, no. 10, pp. 1448–1484, 2019.

[5] M. Fowler and J. Lewis. (2014, March) Microservices a definition of this new architectural term. [Online]. Available: https://martinfowler.com/articles/microservices.html

[6] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm," in *Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 1222–1228.

[7] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "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*, vol. 47, no. 9, pp. 1275–1296, 2017.

[8] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 1, p. 9, 2018.

[9] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the internet of things," in *Proceedings of the 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–6.

[10] T. Vresk and I. Čavrak, "Architecture of an interoperable IoT platform based on microservices," in *Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2016, pp. 1196–1201.

[11] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a smart city internet of things platform with microservice architecture," in *Proceedings of the 3rd International Conference on Future Internet of Things and Cloud*. IEEE, 2015, pp. 25–30.

[12] S. Nastic, M. Vögler, C. Inzinger, H.-L. Truong, and S. Dustdar, "rtgovops: A runtime framework for governance in large-scale software-defined iot cloud systems," in *Proceedings of the 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. IEEE, 2015, pp. 24–33.

[13] K. Vandikas and V. Tsiatsis, "Microservices in iot clouds," in *Proceedings of the 2016 Cloudification of the Internet of Things (CIoT)*. IEEE, 2016, pp. 1–6.

[14] I.-D. Filip, F. Pop, C. Serbanescu, and C. Choi, "Microservices scheduling model over heterogeneous cloud-edge environments as support for iot applications," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2672–2681, 2018.

[15] D. Santoro, D. Zozin, D. Pizzolli, F. De Pellegrini, and S. Cretti, "Foggy: a platform for workload orchestration in a fog computing environment," in *Proceedings of the 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017, pp. 231–234.

[16] F. Faticanti, F. De Pellegrini, D. Siracusa, D. Santoro, and S. Cretti, "Cutting Throughput on the Edge: App-Aware Placement in Fog Computing," *arXiv preprint arXiv:1810.04442*, 2018.

[17] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: architecture, key technologies, applications and open issues," *Journal of network and computer applications*, vol. 98, pp. 27–42, 2017.

[18] M. Slabicki and K. Grochla, "Performance evaluation of CoAP, SNMP and NETCONF protocols in fog computing architecture," in *Proceedings of the NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016, pp. 1315–1319.

[19] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[20] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare internet of things: A case study on ecg feature extraction," in *Proceedings of the 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. IEEE, 2015, pp. 356–363.

[21] Z. Yang, Q. Zhou, L. Lei, K. Zheng, and W. Xiang, "An IoT-cloud based wearable ECG monitoring system for smart healthcare," *Journal of Medical Systems*, vol. 40, no. 12, p. 286, 2016.

[22] S. Tuli, R. Mahmud, S. Tuli, and R. Buyya, "Fogbus: A blockchain-based lightweight framework for edge and fog computing," *Journal of Systems and Software*, vol. 154, pp. 22–36, 2019.