DoSP: A Deadline-Aware Dynamic Service Placement Algorithm for Workflow-oriented IoT Applications in Fog-Cloud Computing Environments

Meeniga Sriraghavendra[1], Priyanka Chawla[1], Huaming Wu[2], Sukhpal Singh Gill[3], and Rajkumar Buyya[4]

[1]School of Computer Science, Lovely Professional University, Phagwara, Punjab, India
{sr.meeniga, priyankachawla.cse}@gmail.com

[2]Center for Applied Mathematics, Tianjin University, Tianjin 300072, PR China
whming@tju.edu.cn

[3]School of Electronic Engineering and Computer Science, Queen Mary University of London, UK
s.s.gill@qmul.ac.uk

[4]Cloud Computing and Distributed Systems (CLOUDS), Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia
rbuyya@unimelb.edu.au

Abstract. The next generation Internet of Things (IoT) applications are offering multiple services and run in a distributed heterogeneous environment. In such applications, Quality of Service (QoS) requirements are in jeopardy when the computing operations are only outsourced to the public cloud. For IoT applications a comprehensive framework that supports QoS-aware service placement in a fog computing environment is highly required. It is a challenging task to orchestrate the time critical IoT applications in the fog environment. To alleviate this problem, this paper proposes a novel multitier fog computing architecture called Deadline oriented Service Placement (DoSP) that provides the services both in fog and cloud nodes. This research work proposed a methodology to utilize low cost fog resources while ensuring that the response time satisfies a given time constraint. It uses the Genetic Algorithm (GA) to dynamically determine the service placement in the fog environment. In this work, we used the iFogSim simulator to model DoSP and measured the impact of the service placement technique in terms of service deadline. It has been observed that through the proposed solution, there is a reduction in service execution delay, i.e. approximately 10.19% of the overall response time to the EdgeWard and 2.58% to the cloud-only.

Keywords: Fog computing; IoT; Cloud Computing; Edge Computing; Workflow.

1 Introduction

Cloud computing has been facilitating individuals and organizations by extending access to remote computing resources on subscription basis. In cloud computing, the

servers are located in a remote place and provide computing resources on demand. Servers being in the remote place leads to problems such as high latency, high bandwidth and energy consumption. Because of these disadvantages cloud computing is not ideal for Internet of Things (IoT) applications [1], which are latency sensitive, such as precision agriculture, intelligent transportation, smart homes, smart cities, smart grids, smart healthcare units and smart supermarkets. Providing services in a short time is a major research issue that has been addressed by many researchers in the past few years [4][28][30]. The issue to get the computation closer to the end devices, which produce the computational data, is addressed by fog computing technology. Fog computing is an elongated version of cloud computing with effective network bandwidth utilization, heterogeneous computation, workload distribution, and mobility. Some of the extended features of fog computing are described in the below Table 1.

Table 1 Fog Computing Features

Factor	Description
Low Latency	The best facilities on the edge of the network are provided
Mobility	Disassociates the host identity to the location identity.
Real Time Interactions	Uninterrupted speedy service.
Interoperability	Objects collaborate and communicate with each other during the transmission
Low Energy Consumption	Reducing computation and communication energy consumption.

Resource provisioning can reduce latency and increase compliance of Service Level Agreements (SLAs) [32], which has been investigated in [7], [10], [18], [19], and [25]. However, mere service provisioning cannot achieve the desired performance, which needs to be coupled with service placement strategies as explored in [3], [4], [6], [12], [13], [31], [33] and [34]. So, there is a need for a comprehensive approach that not only takes care of service placement but also enhances Quality of Service (QoS) in the heterogeneous and dynamic fog-cloud computing environment.

The major contributions of this proposed work is as follows: The proposed work provides a comprehensive solution of QoS-aware service placement to realize highly optimized service placement of sequential IoT applications in a fog-cloud computing environment.

1. A Deadline-oriented Service Placement (DoSP) algorithm is proposed, which analyzes the response time of service placement indifferent layers, and determines decisions on placing modules/services of workflow-based IoT application in different layers of fog-cloud architecture.
2. The fog-cloud environment is simulated using iFogSim toolkit and the performance of the service placement algorithm is evaluated. We are able to visualize the service placement and the different layers determine decisions on placing modules/services of multiple IoT applications in different layers of fog-cloud architecture.

The rest of the sections in the paper are organized in the following way: Section 2 presents a motivating scenario for the research carried out. Section 3 reviews the literature on developing and deploying IoT applications, resource allocation and application/service placement in the fog-cloud environments. Section 4 presents the system model which is the basis of the research. The complete methodology used to realize an approach for deadline-aware efficient service placement is provided in Section 5. Section 6 presents the study on simulation that is made with iFogSim simulator. The performance evaluation of our approach and comparison with the state-of-the-art methods is presented. Section 7 concludes the paper along with directions for future work.

2 Motivation Scenario

Fog computing helps benefit IoT devices and applications. This is an important proposition based on which this motivating scenario is conceived. Workflow-based IoT applications in the real world need to desire latency where the edge of the network with storage and computing services (fog computing) plays a crucial role. Extending the cloud computing capability to the edge of the network, and providing a required inductive usage is by Fog Computing. Placement of IoT applications with proper utilization of cloud and fog computing resources is a game changing approach that will have a huge impact on the deployed applications. Figure 1 shows the motivating scenario that can be used to investigate the proposition aforementioned.

As presented in Figure 1, it is evident that a patient respiratory management system is considered as a motivation scenario. Wearable sensors such as sensors are used to detect the vital signs of a patient in relation to the cardiovascular system. The wearable devices equipped with sensors are connected to other digital infrastructure. The sensors provide heartbeat rate, changes in blood volume of the skin and oxygen level carried in both. When sensors provide redundant and irrelevant data, they need to be pre-processed with filtering techniques prior to data analytics. The results of the analysis provide the severity of the problems and accordingly further steps are taken. Sometimes the results reflect an emergency situation where immediate attention is essential. As the application is crucial, in the healthcare domain it is essential to optimize service placement for better latency besides adhering to deadlines if any. It is therefore necessary to evaluate the device modules and make decisions on placements in various layers. Low latency reflects high performance while high latency reflects low performance. This scenario motivates further research on optimal service placement as the availability of the application is indispensable. Every application consists of various modules. The modules of the case study application considered here are described in the following subsections.

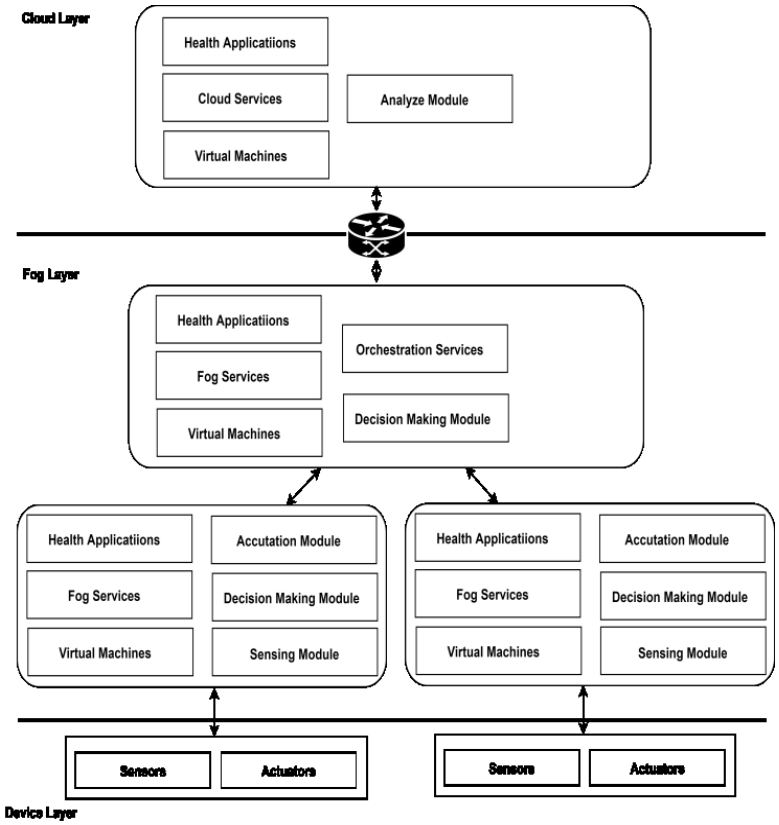


Figure 1 Optimized Application module placement architecture

2.1. Sensing Module

This is the module where actual sensing of data takes place. It is made by the wearable body sensors that are associated with a selected patient. This module is responsible for generating the patient's vital signs related to the respiratory system. Thus, it plays a crucial role in the application, because without this module, other modules would not exist.

2.2. Data Aggregation Module

This module is responsible for data aggregation, which is needed in order to have more meaningful data and also get rid of duplications. It also has a device that converts the analog signal into digital counterparts and needs to be integrated with a microcontroller unit that involves in the data aggregation process, it may include filtration and normalization. The data is aggregated and finally sent to the fog controller node.

2.3. Data Analysis Module

This module is responsible for analyzing data. This is made as per the kind of sensor data, which involves in machine learning algorithms. It is used to test hypotheses. Besides, it needs processing power, storage, and to be placed carefully based on the deadline constraints.

2.4. Decision Making

The analysis module provides the required patterns or interesting facts found in the analysis. These facts are interpreted, and decisions are made by this module. The decisions are pertaining to the real time diagnosis of the patient's vital signs and make the necessary steps. A decision vector holds the resultant information.

2.5. Actuation Module

This module is responsible for turning energy into motion or performing its intended purpose. In general, it converts physical parameters into electrical outputs.

3 Related Work

This segment examines the literature on the location of services in the fog computing world and its associated aspects.

3.1. Building and Deployed Fog-Based IoT Applications

Kochovski and Stankovski [1] focused on dependable edge computing and proposed architecture for constructing smart applications that exploit edge computing in the context of IoT. Their proposed infrastructure includes edge management services, virtual clusters, and smart IoT construction environments. Edge management services perform probing, monitoring, alarming and storing. Edge management services work on top of virtual clusters to support edge computing applications. Pham and Huh [2] considered a cloud-fog computing environment, where fog nodes are used appropriately along with cloud nodes to enhance performance. Giang et al. [11] proposed a distributed data flow approach for building IoT applications to be deployed in fog.

3.2. Resource Allocation for IoT Applications in Fog Environment

It is essential to have resource allocating mechanisms, when the IoT applications are executed in the fog-cloud environment. Skarlat et al. [7] presented an architecture or framework for resource allocation in the fog-cloud environment. Delay sensitive

utilization of fog resources was the aim of the framework. Considering SLAs and QoS, Aazam et al. [10] proposed a framework known as Media Fog Resource Estimation for estimation of resources based on QoS, QoE, Relinquish Rate (RR) and service give-up ratio.

Yousefpour et al. [15] studied the problem of service provisioning in a dynamic and distributed computing environment. To solve the dynamic service placement they have proposed two heuristic algorithms in fog inspite of building a framework to realize it. They found that the framework could decrease SLA violations besides increasing the performance of deployed applications. Aazam and Huh [16] considered many issues with IoT-fog environment due to the mobility of nodes, resource constrained nature of nodes and heterogeneity of the environment. Taking those factors into account, they proposed resource estimation and provisioning framework through fog micro datacenter. Aazam and Huh [17] have researched adaptive resource estimation and cost allocation models for IoT applications implemented in the fog computing area.

An architecture for flexible and adaptive resource allocation in fog computing is proposed by Agarwal et al. [18]. They proposed an algorithm known as Efficient Resource Allocation (ERA) for estimation of resources and allocating the same to IoT applications deployed in fog. On the other hand, Verma et al. [19] focused on reliable services in the fog by proposing load balancing and data replication techniques and reduce dependency on the cloud. Alsaffar et al.[20] suggested an architecture for the dual purpose of the distribution of resources and the allocation of IoT services in a fog computing system. They found that their work is resulted in meeting the QoS of applications and SLAs.

In the context of container based service computing, Tao et al. [23] proposed a dynamic resource allocation algorithm, which is based on scheduling and fuzzy inference system. Their contributions may increase the efficiency of the cluster in terms of average execution time. Ni et al. [24] proposed a model known as Priced Timed Petri Nets (PTPN), which is a resource allocation strategy in fog computing and also a model to support the strategy. Shekhar et al.[25] proposed a dynamic and data-driven cloud and edge system for dynamic resource management in fog in the presence of performance-sensitive applications. Dang and Hoang [26] proposed a framework for efficient scheduling of tasks in the fog to realize expected latency, this framework is known as fog-based region. A joint optimization approach for resource allocation is investigated by Zhang et al. [27] for efficient resource allocation and performance of IoT applications in fog-cloud environments.

3.3. Service Placement in Fog Environment

IoT applications may have micro-services that need to be deployed strategically. Filip et al. [3] studied cloud-edge environments for scheduling micro-services related to IoT applications. They proposed a scheduling architecture for real life utilization of the fog-edge environment effectively. They used a scheduling policy for better task scheduling. IoT applications have multiple modules. Mahmud et al. [4] studied deploying such modules in the fog-cloud environment. They proposed a methodology to fully take advantage of the capabilities of fog nodes in the context of large IoT applications. Their approach resulted in a decrease of latency towards QoS requirements of the

applications. Similarly, Desikan et al. [5] explored latency-aware data processing in the fog-cloud environment. They could reduce the response time and increase the performance of gateways with buffer occupancy efficiencies.
Taneja and Davy [6] proposed an algorithm named Module Mapping, which is meant for deploying modules of IoT application appropriately in fog-cloud resources. As the storage and computation are distributed dynamically, the algorithm was able to reduce latency and improve the capabilities of the cloud-fog environment to withstand SLAs. Skarlat et al. [8] investigated QoS-aware provisioning of services of IoT, on fog resources and found a considerable 35% less cost in terms of execution time while compared with the cloud-based approach. On the other hand, Mahmud et al. [9] considered Quality of Everything (QoE)-aware placement of services or applications in a fog computing environment. They proposed an application policy that is QoE-aware and prioritizes application placement requests as per the expectations of users.
Souza et al. [12] studied the QoS-aware service allocation problem. They proposed fog-cloud architecture with four tiers. The bottom layer composed of end user devices, the layer above is called the fog layer 1, fog layer 2 is on top of fog layer 1 and the top most layer is the cloud. The cloud has high reach ability but causes high access delay with respect to IoT applications. Fog layer 2 exhibits medium reach ability and medium access delay, while fog layer 1 causes low reach ability and low delay. This scenario is to be exploited by QoS-aware service allocation. FogTorch is a java tool which is proposed by Brogi and Forti [13]. The tool is a latency and bandwidth-aware while deploying the IoT applications in the fog. Brogi et al.[35] have reviewed the existing policies to solve the problem of how to dynamically deploy the application modules in the computational components with qualitative attributes. Abbasi et al.[36] addresses the problem of placement of workloads. The purpose of their study is to minimize the energy consumption and propagation delay of the cloud while processing. In this work, the placement problem is solved by using the NSGA-II algorithm. The work does not focus on the application deadline and the propagation delay among the fog nodes.

3.4. A Qualitative Comparison

Table 2 identifies and compares key elements of related works with DoSP. The comparison attributes are the target system, application type, resource utilization, minimization of response time, application priority and deadline constraint. The insights found in the literature reveal that the existing approaches are good to realize fog-cloud environments with increased performance. However, it is understood that a comprehensive framework that not only takes care of service placement but also QoS enhancement in the presence of the highly heterogeneous environment.

4 System Model

The system model considered for the proposed research provides the layered architecture, in which proposed workplaces the multiple sequential IoT applications with a sense-process-actuate model are executed. In the context of this system model

service placement, QoS-aware service placement approach is explored in this research work.

Table 2 Comparison of DoSP with existing works

Work	Target system	App (Application) type	Resource utilization (same cluster/neighbor cluster)	Minimize response time	App (Application) priority	Deadline constraint	Approach
Kochovskietet al. [1]	Edge	workflow	Same cluster	No	No	No	Test bed
Giang et al. [11]	Fog	workflow	Same cluster	No	No	No	Visual programming tool
Taneja et al. [6]	Fog	workflow	Same cluster	No	No	No	Heuristic Algorithm
Abbasi et al.[36]	Fog	workflow	Same cluster	No	No	No	Heuristic Algorithm
Skarlat et al. [7]	Fog	workflow	Same cluster	Yes	No	No	Linear Programming
Pham et al. [2]	Fog	workflow	Same cluster	Yes	No	No	Heuristic Algorithm
Alsaffar et al. [20]	Fog	workflow	Same cluster	Yes	No	No	Heuristic Algorithm
Gupta et al. [22]	Fog	workflow	Same cluster	Yes	No	No	Heuristic Algorithm
Souza et al. [12]	Fog	workflow	Same cluster	Yes	No	No	Linear Programming
Zeng et al. [29]	Fog	Bag-of-tasks	Same cluster	Yes	No	No	Heuristic Algorithm
Mahmud et al. [9]	Fog	workflow	Same cluster	Yes	Yes	No	Fuzzy logic
Goudarji et al. [34]	Fog	workflow	Same cluster	Yes	No	No	Heuristic Algorithm
Pham et al. [28]	Fog	workflow	Same cluster	Yes	No	Yes	Heuristic Algorithm
Mahmud et al. [4]	Fog	workflow	Same cluster	Yes	Yes	Yes	Heuristic Algorithm
Skarlat et al. [30]	Fog	workflow	Same and Neighbor Cluster	Yes	Yes	Yes	Linear Programming
DoSP (this work)	Fog	workflow	Same and Neighbor Cluster	Yes	Yes	Yes	Heuristic Algorithm

FN is meant for storing arbitrary data required by workflow-based IoT applications and also supports computations. It is said to be the local node or the node that is closer to the deployed IoT application. Thus, fog computing is an enhanced version of of cloud

computing which is especially meant for rendering efficient services to workflow-based IoT applications. As per the facts found in the literature, FCN>FN>CN>NFCN is the thumb rule with respect to the speed of the deployed services of IoT applications. In this context, this system model is used to investigate service placement with QoS awareness strategies in place. The proposed mechanism for service placement has two different phases, i.e., application prioritizing phase and node selection phase. In the prioritizing phase, the services of IoT applications are assigned priorities. In the node selection phase, the suitable node in the fog or cloud is selected for the placement of given services. Based on the objective function, the node selection is carried out.

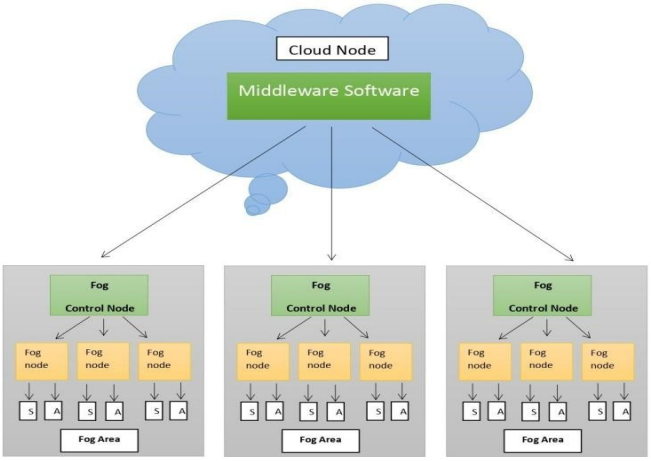


Figure 2 System model

4.1. Application Prioritizing Phase

In this phase, the requests coming from different applications for service placement are subjected to prioritization. Towards this end, based on the distance between the deadline of an application and the deployment time of an application we prioritize the applications, which is defined as follows:

$$\text{prioritization} = \text{Deadline}_{\text{App}_i} - \text{Deptime}_{\text{App}_i} \quad (1)$$

Where $\text{Deadline}_{\text{Appi}}$ represents application deadline, and $\text{Deptime}_{\text{Appi}}$ represents waiting time for deployment. $\text{Deptime}_{\text{Appi}}$ is associated with the time waited by application prior to assignment of it to suitable computing resources.
It is always a better practice to initially assign the applications that have high waiting time based on the deadline to the fog resources. As the deadline cannot be violated, it needs to be made so. Allocating the shortest computation applications first to fog resources may lead to starvation, hence it is not desirable.

4.2. Node Selection Phase

In order to deploy services of an application, it is essential to have certain constraints that can help to achieve optimal performance.

Constraint 1: In addition to storage, this constraint is related to RAM and CPU (important computer resources). These resources are to be considered as the fog resources allocation should not exceed these available resources.

$$\sum_{App_i}^{App} \sum_{AppMod_i}^{App_i} \mathrm{Res}_{\mathrm{AppMod_i}} . \mathrm{Place}^{\mathrm{fn}}_{\mathrm{AppMod_i}} \leq \mathrm{Avail}^{\mathrm{Res^{fn}}}, \forall \mathrm{fn} \qquad (2)$$

Where fn={FN,FCN,NFCN}denotes fog node in a fog cluster, Res={CPU,MEM,STORAGE}denotes required resources, Place denotes service placement and Avail denotes available resources.

Constraint 2: This condition is related to expected response time of the applications. No application should violate its deadline, therefore:

$$\mathrm{Resptime}_{\mathrm{Appi}} \leq \mathrm{Deadline}_{\mathrm{Appi}} , \forall_{\mathrm{Appi}} \qquad (3)$$

Where $\mathrm{Resptime}_{\mathrm{Appi}}$ represents application response time and $\mathrm{Deadline}_{\mathrm{Appi}}$ represents application deadline.

Estimating application response time

In a deadline-aware approach, it is important to estimate application response time. Therefore, the response time can be estimated by:

$$\mathrm{Resptime}_{\mathrm{Appi}} = \mathrm{TotDeptime}_{\mathrm{Appi}} + \mathrm{Exectime}_{\mathrm{Appi}} \qquad (4)$$

$$\mathrm{Exectime}_{\mathrm{Appi}} = \mathrm{Makespan}_{\mathrm{Appi}} + \mathrm{Comm}_{\mathrm{Appi}} \qquad (5)$$

Where $\mathrm{TotDeptime}_{\mathrm{Appi}}$ denotes deployment time, $\mathrm{Exectime}_{\mathrm{Appi}}$ denotes execution time, $\mathrm{Makespan}_{\mathrm{Appi}}$ denotes makespan time, and $\mathrm{Comm}_{\mathrm{Appi}}$ denotes communication time.

$\mathrm{Deptime}_{\mathrm{Appi}}$ considers elapsed time prior to the placement of each service correctly. The placement is done either in cloud or fog based on runtime situations. In fact, $\mathrm{TotDeptime}_{\mathrm{Appi}}$ represents elapsed time and also the expected deployment time. It is made when service $\mathrm{AppMod_i} \in \mathrm{App_i}$ is actually subjected to propagation to nearest colony. The execution time denoted as $\mathrm{AppMod_i} \in \mathrm{App_i}$ reflects the time required for execution of all services of given application plus the amount of communication needed among services. Therefore, the total execution time $\mathrm{AppMod_i} \in \mathrm{App_i}$ is computed as the communication time $\mathrm{Comm}_{\mathrm{Appi}}$ and the sum of makespan $\mathrm{Makespan}_{\mathrm{Appi}}$. In the link, delays reflect communication time in the process of service placement in different areas as illustrated in the architecture.

4.3. IoT Application Placement Flow in Fog Environment

Figure 3 shows a fog controller node receives a service placement request from the user. Then, it identifies the required resources for the requested IoT application. It also takes deadline associated with the service to be placed. Keeping deadline in mind, the resources on fog nodes are analyzed. Then identify fog nodes that can meet service deadline. Afterward, the deadline-deployment time is calculated and verified. If it is low, the service is placed in the fog node (or) fog controller node (or) cloud. If the time is high, then it is forwarded to NFCN and checks the resource availability and QoS expectation. If it is satisfied, the application modules are placed in NFCN, otherwise, they are forwarded to the public cloud where the service is placed.

5 Proposed Methodology

Goal of this proposed work is to develop a comprehensive framework that supports QoS-aware service placement of IoT based applications in a fog based computing environment. The significance of this methodology lies in the context of emerging IoT applications in different domains like healthcare, military and the need for fog computing to accommodate the same besides exploiting the technologies like cloud computing. According to the research objectives mentioned in Section 4, the proposed methodology needs to be divided into multiple sections.

5.1. Overview of the proposed work

This section provides an overview of the proposed work. The framework presented in Figure 4, which provides QoS-aware service placement for ensuring that the placement of services is highly optimized to improve performance. This kind of methodology can help in placing services in a fog computing environment with efficient decision making related QoS requirements of a given IoT application. Workflow-based IoT application(s) is given as input. Then, the proposed system is responsible for analyzing its requirements and executing an algorithm to have an effective service placement that considers the deadline of services. The service placement achieves satisfying deadlines. Here in the proposed work, no service layer agreements are considered, for simplicity.

5.2. Functional Details of the Proposed Work

The proposed architecture for deadline aware service placement is as shown in Figure 5, which is elaborated with functional details. On taking one or more workflow-based IoT applications, the proposed architecture performs a series of activities to ensure that the service placement is done according to the requirements of the application. From the IoT application, the modules of the application are extracted. For each module/service, the details obtained include a number of modules to be placed, the sequence of the modules and the deadline associated with each module. Afterward, the requirements for each module are analyzed. This is nontrivial as each module needs

different resources such as memory, processing power, and bandwidth. Moreover, this requirement for a given service may be fulfilled in FN, FCN, NFCN or CN. Resource availability is then verified to know the information about all nodes and their available resources. The resource availability information is then utilized to make a decision on node selection for a given service or module. Once node selection is made based on the latest analysis of the resource availability considering the deadline associated with the service, the services are finally placed in a selected layer and in selected nodes.

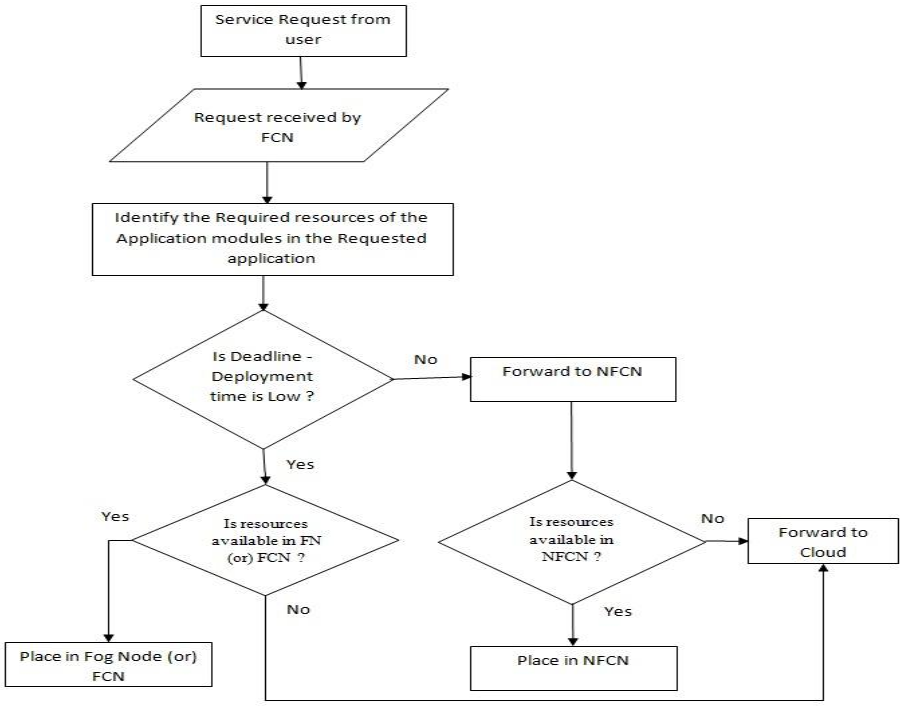


Figure 3 Flowchart of Resource Provisioning for IoT application in Fog based computing environment

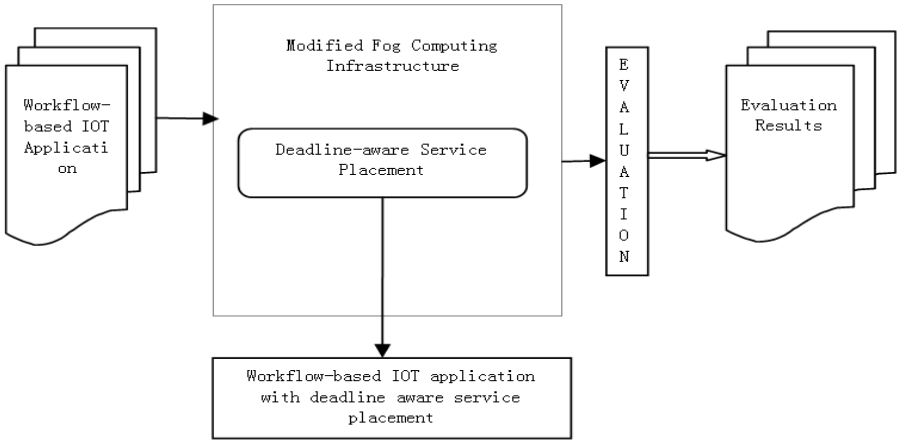


Figure 4 Overview of the framework showing the proposed research

5.3. Deadline-oriented Service Placement Algorithm (DoSP)

Deadline-oriented Service Placement (DoSP) algorithm is designed to analyze the fog environment for a given deadline requirement of a workflow-based IoT application and make well informed decisions on the application or its modules placement in the fog environment. The proposed DoSP algorithm is presented in Table 3.

The proposed algorithm is part of Deadline-aware service placement module in Figure 4 and Node selection module in Figure 5. It is based on the concept of Genetic Algorithms (GA). When using a genetic algorithm, the first thing to be considered is how to model the solution.

Every operator inside the GA will work using that model. The size of the vector considered is equal to the number of services that will be used in the algorithm, i.e., the number of services requested to be executed. The vector is the placement plan which places the application modules. If the vector[i] = 2, then i^{th} module is placed in node 2. Hence, the i^{th} module is placed in the computation node value. We consider this vector (the placement plan) as the chromosome. As said before, it will be in the form of a vector of integers. Possible integers are IDs of fog cells, cloud, closest neighbor, fog orchestration control node. The IDs considered as follows, 0 for Cloud, 1 for FCN, 2 for NFCN, 3 to remaining as FN. The number of possible integers is assigned to the variable “number of Placement Locations”

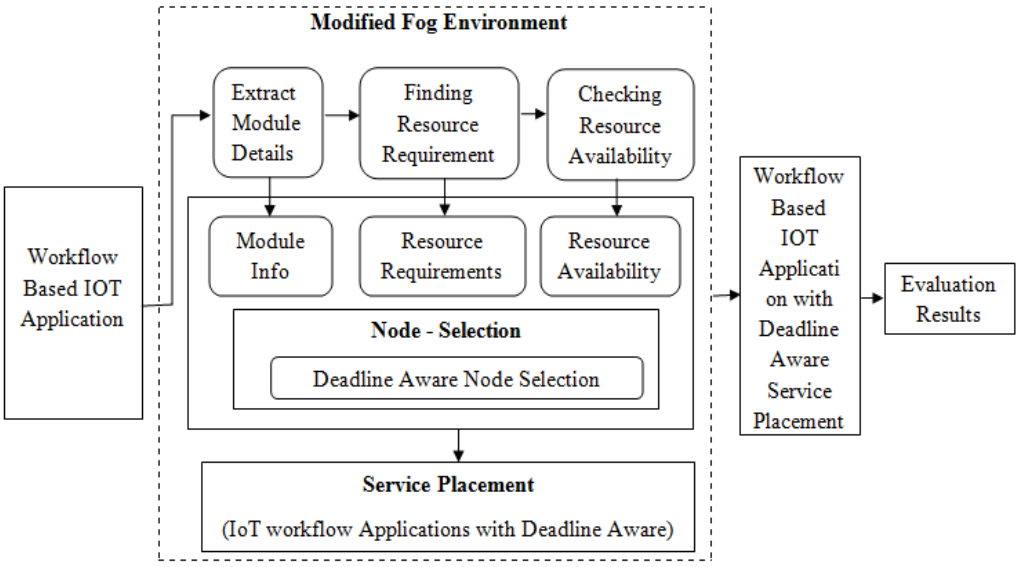


Figure 5 Functionalities of the proposed architecture

After we assign values to variables "Chromosome Length" and "Number of Placement Locations", we need to create the first generation. One of the segments of the GA is the population size that needs to be predefined. With this number, we will go ahead and create many chromosomes (solutions) to form the first generation. There are a lot of different ways how we can create a chromosome, i.e., fill a vector of integers. One way is to randomly put numbers from a set of possible integers to the vector. A new chromosome is created at this stage. All the generated chromosomes should satisfy the state check. A state check is an operation which checks whether the given plan is successfully placed. A module can be successfully placed firstly, when the required MIPS is less than available MIPS secondly, the sense and actuation modules are placed only in FN, and finally process modules are not allowed to place in FN.

After a new chromosome has been created, we calculate its fitness to see how good it really is, and then we add it to the population. The fitness of the chromosome depends on how far the app is away from the deadline. If the application is closer to the deadline then a high penalty is added or else a low penalty is added. Here we need to minimize the fitness value. Minimum fitness chromosome is the best possible placement plan. We assign the fittest chromosome from the population to the variable "fittest Chromosome". If that chromosome satisfies all the constraints, we return it, and this is done. If that is not the case, we will need to use the rest of the algorithm. First, we define how many crossovers will be executed. "Crossover" is another operator inside the genetic algorithm which is used.

Another segment of the genetic algorithm is the number of generations, which is also user defined. In each generation, we used a GA operator "selection" to get the best(fittest) chromosome. The fittest chromosome is, in this case, the one that has the biggest fitness among the selected ones. So, we are not looking at the fittest chromosome directly from the population. The selection part of the algorithm can be

implemented in many ways. One way is to use the Tournament selection. Tournament selection is the type of selection with two steps.

Table 3 Deadline-oriented Service Placement (DoSP) Algorithm

Algorithm 1: Deadline-oriented Service Placement (DoSP)

Inputs: No. of applications, No. Application modules, CPU, Memory Storage, make-span time, deployment time, population.

Output: Efficient and deadline-aware service placement

```
// build the first generation
Chromosome Length = total No. of applications * No. Application modules;
Number Of Placement Locations = FN or CN or NFCN or FCN;

for i = 1 to population Size do
        new Chromosome = create Chromosome(Chromosome Length, Number Of Placement Locations);
         if(state_check(Chromosome))
             calculate fitness of new Chromosome;
             add new Chromosome to population;
             i++
end for
for i = 1 to generation Limit do
        fittest Chromosome = get Fittest Chromosome(population);
  if fitness of fittest Chromosome > 0 then
                return fittest Chromosome;
        add the current population to the temporary population;
   number Of Chromosomes Used For Crossover = generation Size * defined Crossover Percentage;
        number Of Crossovers = number Of Chromosomes Used For Crossover / 2;
// two chromosomes are involved in each crossover
        for j = 1 to number Of Crossovers do
                parent Chromosomes = select Chromosomes(population); // selection
                children Chromosomes = mate Chromosomes(parent Chromosomes); // crossover
                state_check(children Chromosomes)
          state_check(mutate Chromosomes(children Chromosomes)); //mutation and state check
                calculate the fitness of the new chromosomes;
                add the new chromosomes to temporary population;
        end for
        add elite chromosomes from temporary population to the new population;
        add remaining best chromosomes from temporary population to the new population;
end for
```

The first step is to round all the solutions from the population into a “circle”, and depending on the fitness of each solution, it will occupy some part of that circle. The better is the fitness, the greater the percentage of the circle. The second step is to return the fittest out of the selected solutions. This selection is used to get both parents. There are different ways of how a crossover can be implemented.
One way is the uniform crossover. In this crossover, genes (parts of the chromosome) are uniformly selected between both parents to form a child chromosome. Another genetic algorithm parameter is the fixed mixing ratio inside the uniform crossover, which tells us how much of the genes will come from each parent. We are actually creating two child chromosomes with crossover. Those chromosomes are completely opposite. If the first one uses the first gene from the first parent, the second one will use the first gene from the second parent, etc. Here in the proposed algorithm we have used uniform crossover. The resultant chromosome is the valid chromosome which satisfies the state check. A mutation can happen after crossover. The mutation is another genetic algorithm operator. It is rarely used, but it is a tool that helps a lot when we want to have a diverse population. In the mutation operation, we select ‘m’ number of random locations, and we stuff each location with a new node value that satisfies the state check. After the mutation, we calculate the fitness of the new chromosomes and store them (chromosomes) into a temporary and constant list of population. In that temporary list, we also put the whole current population. After this, we do all the crossovers and place all the new chromosomes to the temporary population, we perform elitism. Elitism has a percentage, for example, 20%, which means that 20% of the best chromosomes from the current population will automatically go to the next generation. The rest will be selected from the temporary population by sorting them (chromosomes) based on the fitness value and choosing the best 80%from them. This depends on when the user-defined algorithm stops. When the number of applications and nodes increase the time complexity will increase, this is observed while using genetic algorithms. The asymptotical time consumption or the complexity of the proposed algorithm is $O(nlogn + IPS2)$, where n indicates no. of applications for sorting, I is the no. of iterations, P indicates population size and S implies no. of sub solutions.

6 Performance Evaluation

6.1. System Setup and Parameters

As found in [3], [6], [8] and [10], simulation studies are widely used for fog computing to evaluate resource management policies. Here, we use a fog computing simulation toolkit named iFogSim [22], which is found to be a suitable framework for fog computing research as it is used by a number of research works like [14],[20],[27] and [31].
The environment is important for conducting experiments. A summary of the environment used for the empirical study in this research work is given in Table 4.
The simulation study has been carried out using iFogSim with JDK 1.8. Our simulation framework has an API to visually model the fog computing infrastructure. It has provisions to model the cloud, fog nodes, fog controller nodes, gateways, sensors, and

actuators. The organization of the network is as follows: one ‘Fog Orchestration node’, ten ‘Fog Nodes’ controlled by a ‘Fog Controller Node’, one ‘Neighbor Controller Node’, and a ‘Cloud’. The processing capability of the node can be set with a certain capacity that is measured in Millions of Instructions per Second (MIPS). The iFogSim supports drag and drop features that help users to have intuition in design and understanding.

Table 4 Simulation setup and its configuration

Processor	Intel Core i5-2430 CPU,2.40GHz
Memory	8 GB
Simulator	iFogSim
Operating System	Windows 7 Professional
Topology model	Hierarchy

Figure 6 shows the fog infrastructure modeled for the proposed simulation study.

Table 5 Characteristics of the fog controller node and fog node

Parameter	Fog Controller node	Fog node
Processing rate	1000 MIPS	250 MIPS
Memory	512 MB	256 MB
Storage	8 GB	4 GB

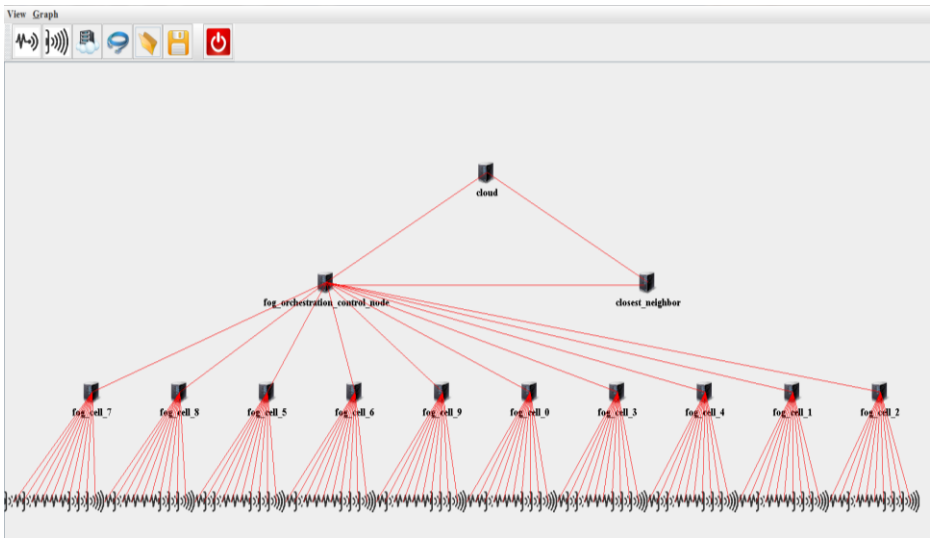


Figure 6 Fog infrastructure modeled using iFogSim for the simulation study

Each fog cell has a provision for different nodes. Once the modeling is made, the nodes are configured appropriately. In the simulation study, each node mimics a real world node with equivalent characteristics. Therefore, it is essential to configure them as required. How fog controller and fog nodes are configured for different parameters is shown in Table 5. The parameters include processing rate, memory and storage details. As shown in Table 6, there are other parameters needed for simulation. These parameters are configured in terms of the communication link delays (in seconds). This makes it easier to observe and measure the results in the simulation study. When the service placement scenario is simulated, observation of various parameters is essential as those parameters are used in the real world fog computing-networks.

Table 6 Different parameters and their corresponding communication link delay in seconds

Parameter	Communication link delay (sec)
Fog controller node-Cloud	9
Fog controller node-Neighbor controller node	0.5
Fog controller node-Fog node	0.3

The communication link delay for Fog controller node–Fog node link is set to 0.3 seconds. The communication link delay for Fog controller node–Neighbor controller node link is set to 0.5. The communication link for Fog controller node–Cloud is configured to have 9 second delay.

Once such configurations have been done, it is important to take care of service modules as well. They need different resources like CPU, memory, storage, and makes-pan. These configurations are listed in Table 7. These details are provided for the well-informed simulation study. For instance, the actuate module is set to have 50 CPU (MIPS), 20 MB main memory and 10 MB storage and 0.50 makespan time in seconds.

Table 7 Required resources for application modules

Service module	CPU (MIPS)	Memory (MB)	Storage (MB)	Makespan (sec)
Sensing module	50	30	10	0.90
Data aggregation module	200	10	30	0.10
Data analysis module	200	20	30	0.10
Decision making module	100	30	30	0.25
Actuate module	50	20	10	0.50

The parameters, deadline and deployment, configurations for different types of applications are as shown in Table 8.

Table 8 Deadline and Deployment time of each application

Application type	Application deadline	Deployment time
Motion	120	60
Video	300	0
Sound	300	60

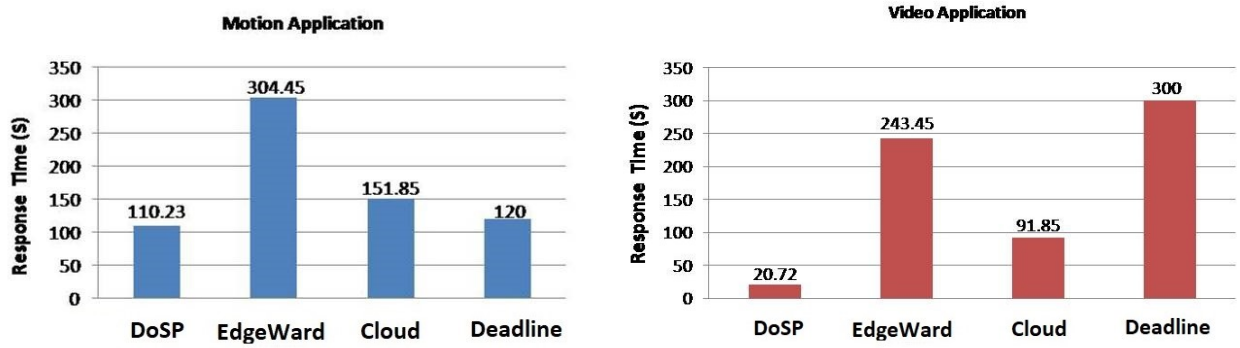
Temp	360	60
Humidity	240	0

For application types associated with motion, video, sound, temp, and humidity, the application deadline and deployment time are determined and presented. The values that were assumed here in Tables 6-8 are based on the average of the previous experimental run.

6.2. Experimental Results

Observations are made on application or its modules placement in fog computing using the experimental setup described in the previous section. The proposed deadline-aware method for service placement methodology is compared with the contemporary methods, e.g., EdgeWard[22] and Cloud Only. EdgeWard is a baseline greedy optimization heuristic, hence it is chosen. EdgeWard places the application modules in the bottom to top approach, based on the availability of resources. In all the cases, Cloud-only places all the application modules in the cloud. Response time is an important observation made at different deadlines. Different workflow-based IoT applications are considered for empirical study. Simulations using iFogSim have resulted in desired insights for the applications such as motion, video, sound, temp, and humidity.

As depicted in Figure 7, the deadline requirement for Motion Application is 120 seconds and Video Application is 300. The horizontal axis provides different service placement approaches, while the vertical axis shows response time in seconds. The results revealed that deadline violation occurred in the case of motion application. In both cases of EdgeWard and Cloud Only methods, the deadline is violated by 184.45 and 31.85 seconds, respectively. The *EdgeWard, Cloud Only and DoSP* approaches could ensure that the time limit for video application is not breached.

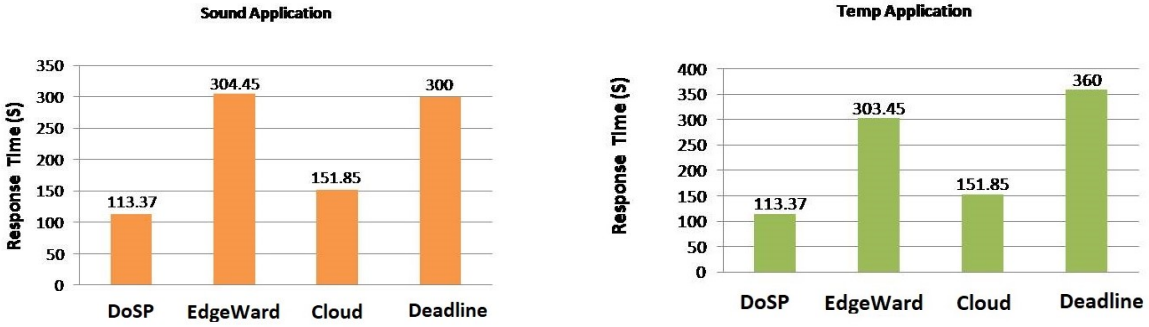


(a) (b)

Figure 7: Performance comparison with (a) Motion and (b) Video applications

As can be seen in Figure 8, it is evident that the results are provided for sound and temp applications. The response time is observed for the IoT applications with deadlines 300 seconds and 360 seconds, respectively. The results revealed that deadline violations

occurred in the case of sound application. The EdgeWard method could not provide service placement ideally as it violated the deadline by 4.45 seconds. In the case of temp application, service placement is made by the three methods without violating the deadline. However, EdgeWard took more response time while DoSP method took the least response time.



(a) (b)

Figure 8 Performance comparison with (a) Sound and (b) Temp applications

As presented in Figure 9, the performance of service placement approaches showed different response times. The proposed DoSP method could honor the deadline. It is the same with Cloud Only approach as well. In the case of EdgeWard method, the deadline is violated by 4.45 seconds. This kind of performance is not acceptable as deadlines in cloud and fog computing scenarios might be associated with SLAs.

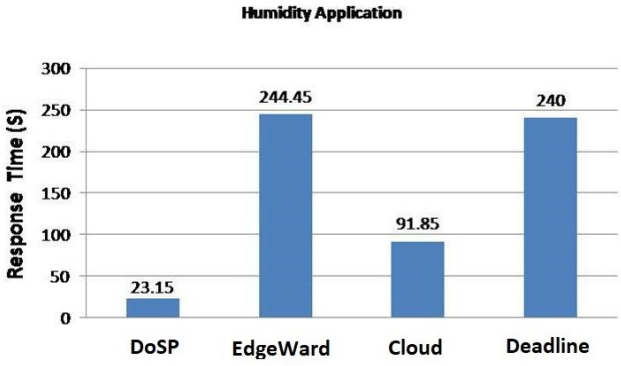


Figure 9 Performance comparison with Humidity application

Response Time against Deadline

Response time or latency is an important metric used for evaluating the proposed DoSP algorithm. The response time against a given deadline is an important empirical observation. As shown in Figure 10, the observations are made in terms of response time against different deadlines. Results of all the applications are presented with the communication distance between the fog controller node and the cloud is 1 second. The

deadlines from 50 to 500 seconds incremented by 50 seconds are provided on the horizontal axis.

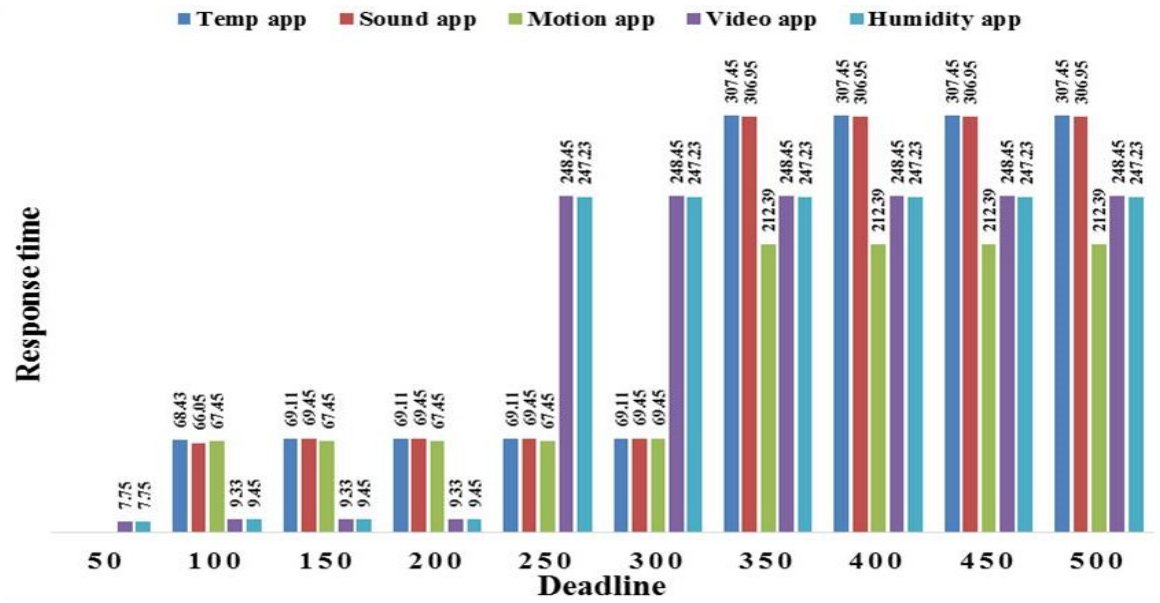


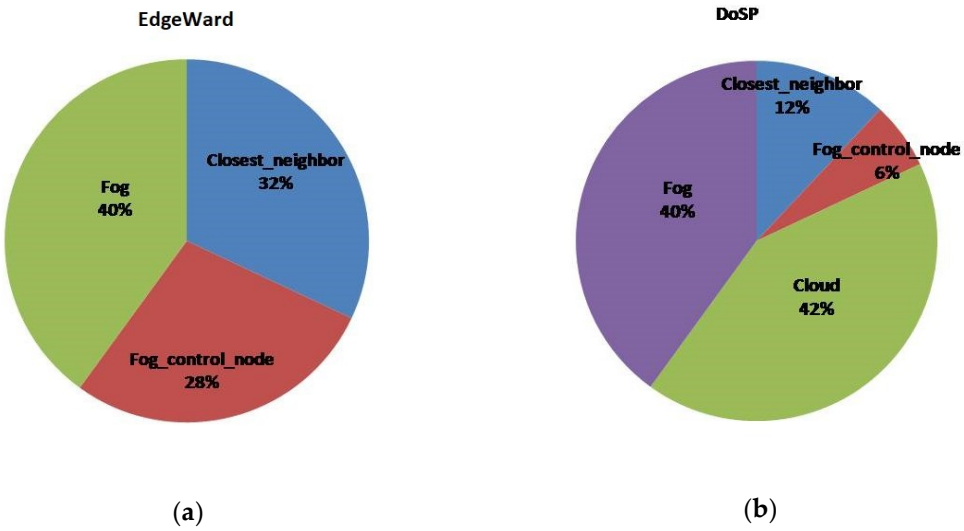
Figure 10 Performance of the DoSP method with respect to all applications

The response time is shown on the vertical axis. When the elapsed time is 50 seconds, both Humidity and Video applications started while other applications started at an elapsed time of 100 seconds. The observations revealed that the response time values can be categorized into 3 levels, namely, lower, medium and high. When services are placed in a fog controller node, response time is less indicating higher performance. In the same fashion, when services are placed in the cloud, the response time is said to be medium, which is greater than that of the fog controller node. The response time is more when services are deployed in the neighbor fog controller node due to deployment delay and extra deployment time to place the service modules. These observations provide the rationale behind the patterns found in response time. The results revealed that the deadline has an impact on the response time. The rationale behind this is that the purpose of the proposed DoSP algorithm is to provide service placement which yields response time that should not violate the given deadline. The results revealed that the proposed method is deadline-aware, and no violations are found with respect to all applications. It does mean that based on the deadline and the resource availability, DoSP has taken appropriate service placement decisions.

Resource Utilization Analysis

Resource availability is analyzed by the proposed DoSP method in order to make service placement decisions. In the EdgeWard method, the sensing and actuating services are placed in various fog nodes in the fog layer. Other activities are known as data aggregation, analysis of data and decision making, are moved to the controller node and propagated to the neighbor fog controller node. In the DoSP, it is different.

The sensing and actuation modules are placed in the fog nodes where as the processing related modules are placed in either fog controller node or in thecloud node.
As shown in Figure 11, based on the resource analysis, 40% of the service placements are made in fog node, which is faster than other nodes. Service placement shows 12% in the closest neighbor node and 6% in the fog controller node.



(a) (b)

Figure 11 Resource utilization for the (a) EdgeWard method and (b) DoSP

The remaining 42% of service placements occur in the cloud. As mentioned earlier, there is a difference in EdgeWard scenarios. It places 40% of services in the fog nodes, 32% in the closest neighbor nodes and 28% in the fog orchestration control nodes.
The Cloud Only approach keeps everything in the cloud. Therefore, it is not able to exploit the fog nodes and fog controller nodes. It has a drawback as the Cloud Only service placement may not meet certain workflow-based IoT applications where quick response time is desired. Cloud is relatively slower than other layers in the fog computing architecture.

7 Conclusion and Future Work

We proposed a Deadline-Aware Dynamic Service Placement (DoSP) algorithm for multiple sequential IoT workflow applications of the same model (sense-process-actuate) and the same number of modules in each application. The applications follow the DDF deployment model. Each application contains the same number of services and each and every service has to be placed on computational nodes.
With the advent of fog computing a platform for the exploitation of available resources at the edge of the network is emerged. The response time of different layers can be stated as $Neighbor\ fog\ controller\ node(NFCN) > Cloud\ node(CN) > Fog\ node(FN) > Fog\ controller\ node(FCN)$. The proposition is interesting as it can speed up IoT services

deployed in fog computing. The proposed work investigates the proposition of the hypothesis aforementioned. It has architecture with multiple layers, i.e., device layer, fog layer, and cloud layer. An evaluation of DoSP algorithm is carried out using iFogSim simulator. The simulation results revealed that the proposed algorithm offers better performance over the approaches like EdgeWard, Cloud Only. DoSP performed well as its placement strategy differs to optimize performance in terms of reducing response time in the presence of strict deadlines associated with SLAs. This conclusion is made using only five workflows based IoT applications but this may not be limited to five and can be extended. The values that were assumed are based on the average of the previous experimental run, which are not extracted from any real-time dataset. In the future, we plan to work on the real-time dataset and investigate energy efficiency for optimization of service placement along with the deadlines considered. In further study a hyper-heuristic approach can also be used for better optimization. Container virtualization along with device mobility in a fog computing environment would also be considered.

References

1. P. Kochovski and V. Stankovski, "Supporting smart construction with dependable edge computing infrastructures and applications," Automation in Construction, vol. 85, pp. 182–192, 2018.
2. X.-Q. Pham and E.-N. Huh, "Towards task scheduling in a cloud- fog computing system," in 2016 18th Asia-Pacific network operations and management symposium (APNOMS), pp. 1–4, IEEE, 2016.
3. I.D. Filip, F. Pop, C. Serbanescu, and C. Choi, "Micro services 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.
4. 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, pp. 1–21, 2018.
5. K. S. Desikan, M. Srinivasan, and C. S. R. Murthy, "A novel distributed latency-aware data processing in fog computing-enabled IoT networks," in Proceedings of the ACM Workshop on Distributed Information Processing in Wireless Networks, pp. 1–6, 2017.
6. M. Taneja and A. Davy, "Resource aware placement of IoT application modules in fog-cloud computing paradigm," in 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 1222–1228, IEEE, 2017.
7. O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for IoT services in the fog," in 2016 IEEE 9th international conference on service-oriented computing and applications (SOCA), pp. 32–39, IEEE, 2016.
8. O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-aware fog service placement," in 2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC), pp. 89–96, IEEE, 2017.
9. R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of Experience (QoE)-aware placement of applications in fog computing environments," Journal of Parallel and Distributed Computing, vol. 132, pp. 190–203, 2019.
10. M. Aazam, M. St-Hilaire, C.-H. Lung, and I. Lambadaris, "MeFoRE: QoE based resource estimation at fog to enhance QoS in IoT," in 2016 23rd International Conference on Telecommunications (ICT), pp. 1–5, IEEE, 2016.

11. N. K. Giang, M. Blackstock, R. Lea, and V. C. Leung, “Developing IoT applications in the fog: A distributed dataflow approach,” in 2015 5th International Conference on the Internet of Things (IOT), pp. 155–162, IEEE, 2015.
12. V. B. C. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, and G. Tashakor, “Handling service allocation in combined fog-cloud scenarios,” in 2016 IEEE international conference on communications (ICC), pp. 1–5, IEEE, 2016.
13. A Brogi and S. Forti, “QoS-aware deployment of IoT applications through the fog,” IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1185–1192, 2017.
14. B Y. Lin and H. Shen, “Cloud fog: Towards high quality of experience in cloud gaming,” in 2015 44th International Conference on Parallel Processing, pp. 500–509, IEEE, 2015.
15. A Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, and J. P. Jue, “FogPlan: a lightweight QoS-aware dynamic fog service provisioning framework,” IEEE Internet of Things Journal, vol. 6, no. 3, pp. 5080–5096, 2019.
16. M. Aazam and E.-N. Huh, “Dynamic resource provisioning through fog micro datacenter,” in 2015 IEEE international conference on pervasive computing and communication workshops (PerCom workshops), pp. 105–110, IEEE, 2015.
17. M. Aazam and E.-N. Huh, “Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT,” in 2015 IEEE 29th International Conference on Advanced Information Networking and Applications, pp. 687–694, IEEE, 2015.
18. S. Agarwal, S. Yadav, and A. K. Yadav, “An efficient architecture and algorithm for resource provisioning in fog computing,” International Journal of Information Engineering and Electronic Business, vol. 8, no. 1, p. 48, 2016.
19. S. Verma, A. K. Yadav, D. Motwani, R. Raw, and H. K. Singh, “An efficient data replication and load balancing technique for fog computing environment,” in 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), pp. 2888–2895, IEEE, 2016.
20. A.A.Alsaffar, H.P.Pham, C.-S.Hong, E.-N.Huh, and M.Aazam, “An architecture of IoT service delegation and resource allocation based on collaboration between fog and cloud computing,” Mobile Information Systems, vol. 2016, 2016.
21. N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, “ENORM: A framework for edge node resource management,” IEEE transactions on services computing, 2017.
22. H. Gupta, A. VahidDastjerdi, 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.
23. Y.Tao, X.Wang, X.Xu, and Y.Chen, “Dynamic resource allocation algorithm for container-based service computing,” in 2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS), pp. 61–67, IEEE, 2017.
24. L. Ni, J. Zhang, C. Jiang, C. Yan, and K. Yu, “Resource allocation strategy in fog computing based on priced timed petri nets,” IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1216–1228, 2017.
25. S. Shekhar and A. Gokhale, “Dynamic resource management across cloud-edge resources for performance-sensitive applications,” in 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 707–710, IEEE, 2017.
26. D. Hoang and T. D. Dang, “FBRC: Optimization of task scheduling in fog-based region and cloud,” in 2017 IEEE Trustcom/BigDataSE/ICESS, pp. 1109–1114, IEEE, 2017.
27. H. Zhang, Y. Xiao, S. Bu, D. Niyato, F. R. Yu, and Z. Han, “Computing resource allocation in three-tier IoT fog networks: A joint optimization approach combining stackelberg game and matching,” IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1204–1215, 2017.
28. X.-Q. Pham, N. D. Man, N. D. T. Tri, N. Q. Thai, and E.-N. Huh, “A cost-and performance-effective approach for task scheduling based on collaboration between cloud and fog

computing," International Journal of Distributed Sensor Networks, vol. 13, no. 11, p. 1550147717742073, 2017.
29. D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," IEEE Transactions on Computers, vol. 65, no. 12, pp. 3702–3712, 2016.
30. Skarlat O, Nardelli M, Schulte S, Dustdar S (2017) Towards QoS aware fog service placement. In: 1st IEEE international conference on fog and edge computing (ICFEC), 2017. IEEE, Madrid, Spain, pp 89–96.
31. S.S. Gill, P. Garraghan, and R. Buyya, ROUTER: Fog Enabled Cloud based Intelligent Resource Management Approach for Smart Home IoT Devices, Journal of Systems and Software, 154, 125-138, 2019.
32. H. Wu, W. J. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 7, pp. 1464–1480, 2019.
33. H. Wu and K. Wolter, "Stochastic analysis of delayed mobile offloading in heterogeneous networks," IEEE Transactions on Mobile Computing, vol. 17, no. 2, pp. 461–474, 2017.
34. M. Goudarzi, H. Wu, M. Palaniswami, and R.Buyya, An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments, IEEE Transactions on Mobile Computing (TMC), 2020.
35. A. Brogi, S. Forti, C. Guerrero and I. Lera, "How to Place Your Apps in the Fog-State of the Art and Open Challenges", arXiv preprint arXiv:1901.05717, 2019.
36. M.Abbasi, E. Mohammadi Pasand, & M.R. Khosravi, "Workload Allocation in IoT-Fog-Cloud Architecture Using a Multi-Objective Genetic Algorithm," J Grid Computing 18, 43–56 (2020). https://doi.org/10.1007/s10723-020-09507-1.