Future Generation Computer Systems 93 (2019) 156–169

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Fog-Based Data Distribution Service (F-DAD) for Internet of Things (IoT) applications

Firat Karatas *, Ibrahim Korpeoglu

Department of Computer Engineering, Bilkent University, Ankara, Turkey

HIGHLIGHTS

- Proposes hierarchical architecture of IoT nodes, fog and cloud computing data centers.
- Uses data classification to efficient store data for multiple applications.
- Presents a data-centric data placement linear programming model.
- Gives data placement strategies for the proposed architecture.

ARTICLE INFO

Article history:
Received 28 June 2018
Received in revised form 17 September 2018
Accepted 22 October 2018
Available online xxxx

Keywords:
Internet of Things
Fog computing
Data placement
Cloud computing
Network topology
Data management

ABSTRACT

With advances in technology, devices, machines, and appliances get smarter, more capable and connected to each other. This defines a new era called Internet of Things (IoT), consisting of a huge number of connected devices producing and consuming large amounts of data that may be needed by multiple IoT applications. At the same time, cloud computing and its extension to the network edge, fog computing, become an important way of storing and processing large amounts of data. Then, an important issue is how to transport, place, store, and process this huge amount of IoT data in an efficient and effective manner.

In this paper, we propose a geographically distributed hierarchical cloud and fog computing based IoT architecture, and propose techniques for placing IoT data into the components, i.e., cloud and fog data centers, of the proposed architecture. Data is considered in different types and each type of data may be needed by multiple applications. Considering this fact, we model the data placement problem as an optimization problem and propose algorithms for efficient and effective placement of data generated and consumed by geographically distributed IoT nodes. Data used by multiple applications is stored only once in a location that is efficiently accessed by applications needing that type of data. We perform extensive simulation experiments to evaluate our proposal and the results show that our architecture and placement techniques can place and store data efficiently while providing good performance for applications and network in terms of access latency and bandwidth consumed.

1. Introduction

The idea of enabling communication between any kind of devices has led to the emergence of Internet of Things (IoT), which is a concept of connecting all kinds of devices with each other via the Internet. The idea has started by designing RF identifiers for small devices and things using radio frequency identification (RFID) technology and monitoring the states of tagged devices, and continued with the developments in wireless sensor networks (WSN) [1]. Nowadays, the types of nodes constituting IoT are increasing every day and the total number of these connected devices is growing enormously, which enables building a lot of different and interesting applications [2]. A lot of studies in literature explain IoT technologies, future directions and research challenges [3–6].

One of the earlier and popular IoT applications among many interesting and potential applications of use cases is crowd-sensing [7]. Since the number of potential applications is growing steadily, a classification is needed to see the trends and analyze the requirements, and Scully [8] does this by classifying all IoT applications into ten different groups. These applications are reviewed and explained in various papers such as [9–11], and one of the common problems of these applications is handling big data. Since the number of IoT nodes can increase enormously, the volume of data generated and consumed by these nodes can increase to an unprecedented level. Storing and processing such a big volume of data become a daunting task [12]. This becomes especially challenging when IoT nodes are geographically distributed to a very large region [13].

* Corresponding author.
E-mail addresses: firat.karatas@cs.bilkent.edu.tr (F. Karatas), korpe@cs.bilkent.edu.tr (I. Korpeoglu).

https://doi.org/10.1016/j.future.2018.10.039
0167-739X/

Distributed cloud computing technology is an attractive alternative to store and process such a large volume of data [14]. Cloud computing can be used to assign complicated and resource hungry tasks to capable data centers distributed around the world [15]. New architectures are needed, however, to integrate cloud computing with IoT [16]. Cisco is one of the first companies suggesting the idea of edge and fog computing that extends the cloud computing capabilities closer to the places where data is generated and consumed [17]. In this architecture, besides large cloud computing data centers, there are a lot of smaller data centers distributed in the region of interest providing fog computing capabilities, storing and processing data in the field. Fog computing data centers are small projections of more capable cloud data centers. Applications, research challenges and other issues related to fog computing are reviewed in various articles [18–20].

In this paper, we propose a mixed hierarchical architecture consisting of geographically distributed cloud and fog computing data centers to extend the capabilities of IoT devices, and to store and process large volumes of IoT data. Leaves of the architecture are IoT devices and they are the least capable components of the architecture. The primary service offered by the cloud and fog data centers in the architecture is data-as-a-service (DaaS) [16], which may enable commercially valuable data-driven IoT applications in a cost-efficient manner.

With data-as-a-service, a network consisting of IoT nodes with both cloud and fog computing data centers can create a data market [21]. In such data market, there are data generators, consumers, and marketplaces. Data is provided to the market by various types of sensors, which are less capable IoT nodes, and applications running in smarter IoT nodes consume the generated data. Fog and cloud computing data centers are the marketplaces where data storages and exchanges occur. Therefore, IoT nodes can be considered as data generators, data consumers or both, and data centers are the places where data resides.

We propose the IoT nodes in a certain local region to be considered together with the fog data center(s) near them to constitute an architectural element called Fog Computing Unit (FCU). These units are built by the combination of heterogeneous IoT nodes and one or more fog computing data center(s). In each unit, capabilities of all elements can be shared among others.

A lot of customers, i.e., applications, may share common demands for data and this may become an important optimization factor, since data amounts shared by applications may be quite large. If each application stores the needed data separately to maximize its benefits, which we call the *application-centric approach*, storing and handling such a large amount of data can become very costly and inefficient. This raises the question of how to handle a huge amount of data without disrupting the application requirements.

In this paper for overcoming the problem, we propose that data generated by IoT nodes to be considered in types and each type of data to be stored only in one location that is optimum for multiple applications requiring it, which we call as the *data-centric approach*. We also give methods about where to store different types of data efficiently so that the applications can reach to data with the lowest feasible latency.

To the best of our knowledge, this paper is the first one proposing the consideration of IoT data in different types and at the same time proposing methods to place different types of data in an efficient and effective manner into fog and cloud data centers. While considering the needs of geographically distributed IoT nodes, we also consider how to decrease the storage costs without affecting performance, which is important from the DaaS providers point of view.

The contributions of the paper are summarized as follows:

1. A hierarchical architecture consisting of IoT nodes, fog computing centers, and cloud data centers is proposed.
2. An architecture for classifying data into types for decreasing storage costs without hurting delay requirements is proposed.
3. According to the proposed data classification architectural model, an analytical model for average data access latency of the applications encounter is analyzed and solved by integer programming model.
4. For achieving the best performance in the given analytical model without using linear solvers, a data placement algorithm is proposed which converges to optimal solution both inside Fog Computing Units and in overall mesh network topology while providing better storage efficiency compared to application-centric data placement.

The remaining of the paper is organized as follows. State of the art approaches in IoT and other related studies are given in Section 2. The proposed fog and cloud based hybrid architecture is given in Section 3. Section 4 explains the analytical model and formulation of the data placement in the proposed system architecture. Section 5 gives the details of the placement algorithm. Section 6 presents the experimental results, and finally Section 7 concludes the paper.

2. Related work

Fog computing is an extension of cloud computing into edges of a network, and can be employed in IoT networks to store and process IoT data more effectively and efficiently. We can group the fog enabled IoT related research studies into three major groups: (1) network architectures and use cases, (2) allocation of available resources and (3) data placement strategies.

2.1. Network architecture

The first group consists of architectural and theoretical work done in cloud and fog enabled IoT. In one of these works, Bonomi et al. [13] explain the interplay of data in the combination of IoT and fog/cloud computing paradigms, and give some architectural structures for applications running in IoT nodes. In another work, Zanella et al. [9] describe a proof-of-concept system architecture for a smart city application which is deployed in Padova. Architectural designs vary according to use cases, and Santos et al. [22] explain a hypothetical network structure for smart city applications using 5G network. Most of the network architectures enabling fog computing concept require intelligent gateways and routers in the edges. Aazam and Huh give a good example of smart gateway concept in [23], and Jutila gives an efficient edge router in [24].

2.2. Resource allocation

The second group is resource provisioning or allocation, which is inevitable where resources are limited and lots of customers, users, applications, etc., need to use them. Proper resource allocation is one of the most important problems not only in cloud and fog computing but also in IoT where the capabilities of nodes are limited, and therefore resources should be used efficiently by applications and users. There is a lot of work done on resource provisioning in cloud computing data centers and a lot of papers, such as [25–28], consider energy-efficient allocation of resources.

An important resource in IoT that needs to be used carefully is network bandwidth, which is to be shared by a lot of applications. Angelakis et al. [29] propose an allocation model for heterogeneous resource demands by considering activation and utilization metrics of available network interfaces in IoT devices. Tsai [30] proposes a network resource allocation algorithm for IoT devices,

called SEIRA, based on search economics for exploring solution space in getting close to optimum.

Resource provisioning is also a very important problem in more limited fog computing data centers. Tocze and Nadjm-Tehrani [31] classify resources in fog computing and survey the related research works and issues based on their taxonomy. In their work, they discuss that data and storage mechanisms are not well-studied in literature. Another important shared resource in fog data centers is computational components, and their utilization is considered in some workload allocation studies. Deng et al. [32] model a hybrid fog–cloud computing architecture by dividing the network into four subsystems, and propose a mathematical framework for optimizing workload sharing mechanism among data centers while considering power consumption and delay. Tong et al. [33] present a workload allocation strategy for mobile computing nodes by pushing cloud data centers to the edges hierarchically and naming them as hierarchical edge cloud. They try to allocate available computational resources on data centers used by mobile workloads efficiently. Besides computational resources, I/O interfaces and resources are also limited in fog computing nodes, and Zeng et al. [34] consider these in a fog computing enabled embedded system.

There are also studies focusing not only on one specific type of resource but also considering the general concept of resource sharing in fog computing. Arkian et al. [35] describe MIST, which is a less dense communication scheme for fog computing, and model resource provisioning problem with a mixed-integer nonlinear programming (MINLP) model by considering limited capabilities of fog nodes in mobile crowd-sensing applications. They mainly focus on a specific application and on the reduction of their nonlinear MIST model to a linear one. Skarlat et al. [36] discuss a software-based resource allocation scheme in fog enabled IoT environments with the help of fog colonies. They try to distribute task requests or data among these colonies by using an entity called fog cell. Although fog colonies resemble fog computing units, they do not include IoT nodes and their work does not consider data and applications independently. Yu et al. [37] tackle resource sharing problem mainly focusing on applications. They investigate real-time application provisioning in a fog enabled IoT network with the aim of satisfying applications' quality of service (QoS) requirements such as bandwidth and delay. Although they consider latency encountered by applications in their work, they do not focus on where data resides or how it is placed.

2.3. Data placement

The third group of related work is data placement strategies. Qin et al. [38] discuss the differences between data characteristics of IoT and traditional Internet applications while focusing on the data taxonomy in IoT. They emphasize that in the conventional Internet, data is mainly generated by human beings, but it is generated and consumed by devices in IoT. Since it is easy to distribute these smart and interconnected devices all around the world, data can be generated and consumed by geographically distributed nodes, and as the number of smart devices increases, data placement becomes an important issue. Problems relevant to fog computing are also valid in other more mature domains such as Online Social Networks (OSN) where the users are geographically distributed as well. Yu and Pan [39] handle data placement problem in OSNs by using hypergraph partitioning techniques in geographically distributed data centers and nodes. In their work, they do not consider the capacity limitations of data centers, which is crucial in fog computing enabled IoT networks. Tang et al. [40] discuss the advantages of geographical distribution of fog computing nodes in smart cities for handling the big data generated by IoT nodes in various use cases.

Various other researchers also consider data-centric use cases and placement. Oteafy and Hassanein [41] discuss the importance and advantages of data-centric placement in fog enabled IoT architectures for reducing access latencies, and envision that applications requiring low latency can proliferate. An example of these applications is streaming based traffic monitoring [42]. Publish–subscribe models are also investigated and one model for DaaS on clouds has been proposed in IoT architectures by defining a quality of data metric, which relies on extracting useful and required data in smart city applications [43].

3. Proposed system architecture

3.1. Network architecture

The system architecture that we propose in this paper considers the hierarchical structure of geo-distributed IoT data that is mentioned in the work of Bonomi et al. [13]. In their geo-distributed structure, latency increases as data goes from IoT nodes to cloud computing centers and this becomes a problem for applications requiring low latencies. A lot of IoT applications are inherently geo-distributed, like smart-city applications, and therefore increased latency in such a geo-distributed system is inevitable if proper precautions are not applied. Bearing in mind, in this paper, we propose a hierarchical system and network architecture consisting of IoT nodes and cloud computing data centers, and fog computing centers in between. This is similar to the neighborhood concept in smart cities which includes fast responsive edge computing nodes connected to IoT nodes [40]. In our proposed architecture, we replace the edge nodes by (F)og (C)omputing data centers (FC), which are small projections of cloud data centers and located near to the field. IoT nodes in a region are connected to a fog computing data center located near them and together they form a storage and processing unit called (F)og (C)omputing (U)nit (FCU). The remaining building block of the architecture is (C)loud (C)omputing data centers (CC), as usual, and they are possible candidates for storing and serving data together with fog computing nodes. If the available resources of fog computing data centers are not enough for storing data, the cloud data centers are the ultimate places to store data.

A large volume of data is generated and consumed by IoT nodes. IoT nodes in a region send and receive data to/from their directly connected FC nodes. In an FCU, IoT nodes and the FC node(s) are connected together in a star topology (see Fig. 1). There may be more than one FC in an FCU. All FCUs and CCs are connected together via a mesh logical topology. The main difference between FCUs and CCs is the availability of resources. Since FCs are small projections of CCs and IoT nodes have restricted capabilities, available resources in FCUs are limited, but in CCs resources are assumed as unlimited.

3.2. Data-Centric placement

In Section 3.1 we define the network architecture which only shows the possible places of data, we formulate the data placement problem in this section. The big data generated for possible consumption needs to be first stored in the network efficiently and effectively. It can either be stored regionally in the FCUs or in the CCs. Since a lot of applications may need the same type of data, we can develop clever placement approaches and algorithms for storing data. Instead of storing the needed data for each application separately, it is possible to store it once and distribute it among all needing IoT applications. This requires first the categorization of data. We propose the generated data to be partitioned, i.e., classified into several well-defined types and all data of some certain

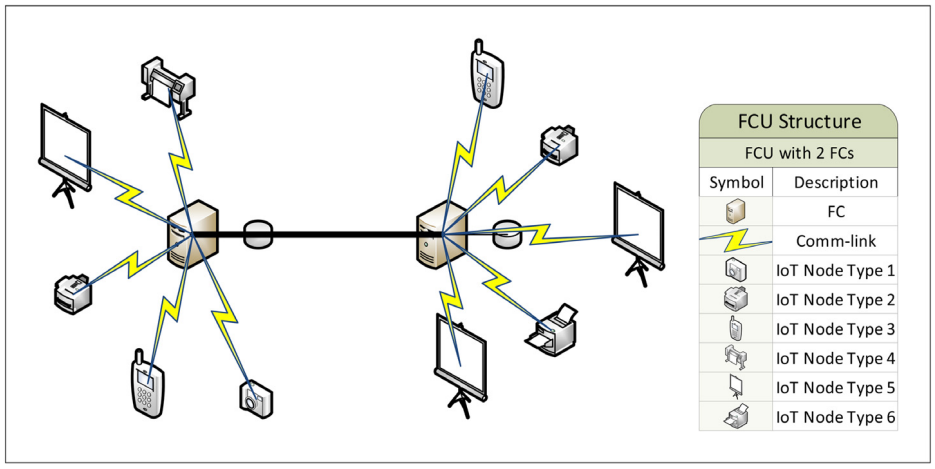

Fig. 1. An Example FCU consists of 2 FCs and 10 IoT nodes.

type to be stored only once and shared by all interested applications. Also, some applications may require several different types of data for doing their jobs correctly. In this case, they access all the required data of different types from the locations where they reside. We call this as *data-centric* approach for placing, storing and using data by multiple applications. We call the other approach where data of some certain type is stored separately for each interested application as *application-centric* data placement. Data is not shared in application-centric approach.

As a motivational example, consider an accident between a car and a pedestrian in a smart city. Smart poles observing the accident can generate health data of the pedestrian and can monitor the traffic nearby. Connected cars in the city can also generate information about the traffic. Let another case be an emergency situation related to a resident in one of the smart homes which is also happening almost at the same time as the traffic accident. Sensors in the home generate vital data of the patient and send it to health services for an emergency rescue. By gathering both health and traffic data, the emergency services can easily optimize available resources, which are ambulances in these cases, and give the task of reaching emergency incident places to the ambulances in the fastest way. In the example, traffic and health data are two different types of data generated by IoT nodes distributed all over the city. Assigning and routing ambulances for two distinct incidents as fast as possible is an application requiring these two different types of data. Now, consider the navigation application which is popular and commonly used in a smart city as the second application running. It also requires traffic data as in the case of health services and if application-centric data placement is used, traffic data will be duplicated. By using the data-centric approach and classifying data into types and placing them accordingly, we can avoid the unnecessary replication of data and reduce the storage costs accordingly.

3.3. Data classifier and data profiler agents

The data-centric approach requires decoupling of data from applications. Using the structural elements defined in [44] with intelligent classification agents running in them, enable the decoupling of data from applications, and make data-centric placement approach possible. IoT nodes are connected to an FC and they are the sources of generated data. An agent process, called (D)ata (C)lassifier (DC), running in all FCs can be used to classify data generated in the network. DC can also monitor the data generation volumes of each data type in each FCU during the classification process. IoT nodes are also the places where applications run and consume data, so their access characteristics to data and their running frequencies need to be profiled for intelligent data placement as well. Another agent called (D)ata (P)rofiler (DP) works as profiling mechanism of applications. DC and DP agents can measure and derive the data characteristics of IoT nodes, and notify the central mechanism for data placement procedures (see Fig. 2). Since IoT nodes are elements of FCUs, outputs of DC and DP agents running in FCs give an idea about the data generation and usage characteristics in FCUs. According to DC and DP agents' outcome, data placement decisions can be done adaptively.

An important benefit of classifying data and storing it once for each class is added flexibility for designing new applications without considering data needs. Designers can use a publish–subscribe mechanism for accessing available data types in the network. From data management perspective, duplication is not needed in the case where data is used by more than one application and this enables reducing storage costs. Regarding that, the data-centric approach can perform better than the application-centric data placement approach dramatically.

In spite of decreasing storage costs in the data-centric approach, however, latencies encountered by applications have to be kept in mind as well. We can achieve this by placing required data to the geographically close places where interested applications run. The issue here is that more than one application may want to use the data of the same type and these applications can be at distinct and far away locations. Another assumption in the architecture is that each application running in IoT nodes needs to access a small amount of data (compared to whole data stored) in the interaction time. Therefore, propagation delay in accessing the needed data of certain type dominates the response time. Effect of the processing delay is negligible.[1] With the help of DP, running frequencies of applications can be compared and for reducing the average latency encountered, the shared data type can be placed near to the application which runs and accesses it more. This reduces data placement to an optimization problem where the average data access latency of applications running in geographically distributed FCUs needs to be minimized without duplicating the data.

[1] The reason is although optical fiber connections are used, a request has to be made from the demanding node and it is transmitted at the speed of light. A small response message is transmitted back to the requested node from the destination which also travels at the speed of light. So the propagation delay becomes "(2 × Distance) / Speed of light", and it is roughly 0.4 ms for a 60 km distance. If small amount of data is assumed to be 1 KB, then its processing time becomes in the order of nanoseconds when a simple RAID 0 architecture used with two SSDs whose read speeds are 500 MB/s with SATA III interface. In today's data centers, much faster SSDs with PCI-Express interfaces can be used and this also reduces the processing times significantly since their read/write speeds are in the order of GB/s.

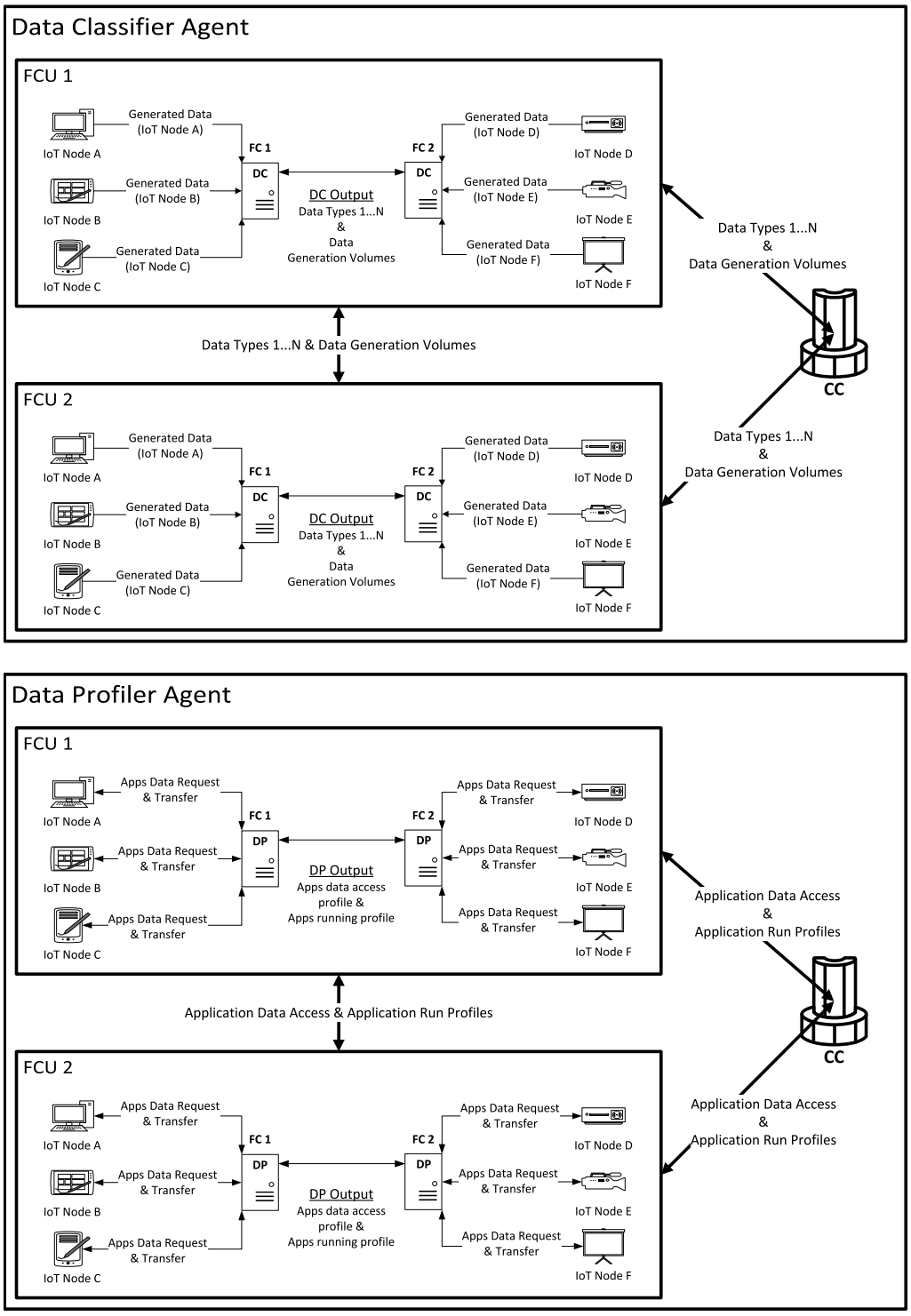

Fig. 2. Data classifier & data profiler agents example scenarios in the proposed architecture.

Under the scope of our model, the mobility of the applications or fast changes that affect the outcomes of DC and DP agents are not considered. Our placement model assumes that all outcomes of agents are available and placement decisions are made accordingly. If significant changes are detected in the outcomes of the agents, placement algorithms have to be rerun.

We formulate this optimization problem and give its linear model in Section 4.

4. Problem formulation

4.1. System model

As mentioned in Section 3, there are three main building blocks of fog supported IoT system: IoT nodes, fog computing data centers, and cloud data centers. In our architecture, all IoT nodes in a local region are connected to one FC node, and there may be more than one FC node in the region. All these IoT nodes and FC nodes in that region are called an FCU. Data is generated and consumed by FCUs.

Generated data can be placed either in FCUs or in CCs. Applications running in FCUs consume data. We consider the data in types and an application may require one or more types of data. Multiple applications can use the same data type but that data type does not have to be handled and stored separately for each application. Applications can share same data types.

If there are more than one FCs in an FCU, we consider them as a single larger FC. Hence, for simplicity, we assume there is one FC serving all the connected IoT devices deployed in a certain region. A geographical area, like a city, has many regions and in each region, there is a different FCU.

Since FCUs are geographically distributed and therefore far away from each other and from CCs, there is a non-negligible latency in the communication of FCUs with each other and with CCs. We assume that this latency is directly proportional with the physical distance and therefore we model the latency between two points as the geographical distance in between.

Our system model has the following parameters:

$\boldsymbol{M}$ → Total number of CCs
$\boldsymbol{N}$ → Total number of FCUs
$\boldsymbol{D}$ → Total number of different data types existing
$\boldsymbol{A}$ → Total number of different applications running
$\boldsymbol{L}$ → Latency matrix showing the latency between any CC or FCU and any other CC or FCU

where $\boldsymbol{L}$ is a $(\boldsymbol{M}+\boldsymbol{N})\times(\boldsymbol{M}+\boldsymbol{N})$ matrix which is directly related to geographical locations of data centers. The placing of the elements into $\boldsymbol{L}$ matrix starts from CCs, for example $\boldsymbol{l_{1,M+1}}$ denotes the latency of the first CC to the FCU numbered as 1. $\boldsymbol{l_{a,b}}$ indicates the latency between data centers **a** and **b** and we define it as:

$$l_{a,b} = \begin{cases} x \in \boldsymbol{R}^{+}, & if \quad a \neq b \\ 0, & if \quad a = b \end{cases}$$

An application does not necessarily run in every FCU. It may either run in a few of them or in all FCUs. Additionally, an application may not always access data or require all types of data. After getting data, an application may spend time for a while to process the gathered data. So, the DP agent running in FCs can measure, for a certain time interval, how often applications run in IoT nodes (i.e., in FCUs) and which data types they access and how often. We denote these frequencies as follows:

$\boldsymbol{AR}$ → Application running frequencies in FCUs
$\boldsymbol{AF}$ → Data access frequencies of applications

where $\boldsymbol{AR}$ is an $(\boldsymbol{N}\times\boldsymbol{A})$ matrix and $\boldsymbol{AF}$ is an $(\boldsymbol{A}\times\boldsymbol{D})$ matrix. $\boldsymbol{AR}$ is normalized according to the most frequently run application.

Every FCU may have different number of IoT nodes and every type of data may not be generated in every FCU. Hence, the amount of data produced may change from one FCU to another. After generation of data, it has to be stored in one of the fog or cloud data centers in the network. We denote the data generation volume of each type of data and where they are placed as follows:

$\boldsymbol{GV}$ → Data generation volume for each type of data in each FCU
$\boldsymbol{DG}$ → Total data generation volume for each type of data
$\boldsymbol{P}$ → Placement matrix where finally data is stored

where $\boldsymbol{GV}$ is a $(\boldsymbol{N}\times\boldsymbol{D})$ matrix, $\boldsymbol{DG}$ is a $(\boldsymbol{D}\times 1)$ vector and $\boldsymbol{P}$ is a $(\boldsymbol{D}\times(\boldsymbol{M}+\boldsymbol{N}))$ matrix. If the elements of these matrices are considered individually, $\boldsymbol{gv_{ki}}$ indicates data generation volume of data type $\boldsymbol{i}$ in FCU $\boldsymbol{k}$, $\boldsymbol{dg_i}$ describes the total data generation volume of data type $\boldsymbol{i}$, and finally $\boldsymbol{p_{ik}}$ indicates whether data type $\boldsymbol{i}$ is placed or not in data center $\boldsymbol{k}$ which may be an FCU or a CC. $\boldsymbol{DG}$ is the transpose of the column sum of the $\boldsymbol{GV}$ matrix, and the elements of $\boldsymbol{DG}$ satisfy Eq. (1).

$$dg_i = \sum_{j=1}^{N} gv_{j,i} \tag{1}$$

$\boldsymbol{P}$ is a binary matrix because partial data placement and replication are not allowed, and it has to satisfy Eq. (2).

$$\sum_{k=1}^{M+N} p_{i,k} = 1, \forall i \in D \tag{2}$$

According to the system architecture described in Section 3, FCs have a limited storage capacity since they are small-scale versions of CCs. Therefore, we have storage capacity constraint for each FCU and we use a variable to denote it. We also use a variable to denote how much of the capacity of an FCU is used:

$\boldsymbol{SC}$ → Storage capacity of FCUs
$\boldsymbol{UC}$ → Used capacity of FCUs

where $\boldsymbol{SC}$ and $\boldsymbol{UC}$ are $(1\times\boldsymbol{N})$ vectors. After placement of all data, we need to have the following Eq. (3) satisfied:

$$uc_i \leq sc_i\,, \quad i \in \{1, 2, \ldots, N\} \tag{3}$$

Referring back to the problem definition described verbally in Section 3, the aim is to minimize the average latency that applications encounter while obtaining the required data from the centers (fog or cloud centers) where data is stored. Table 1 summarizes all the notations used for describing system model. Eq. (4) describes the average latency applications encounter while accessing data.

$$\sum_{i=1}^{A} \frac{\sum_{k=1}^{N} ar_{k,i} \frac{\sum_{y=1}^{D} \mathbf{sgn}(af_{i,y})(\sum_{z=1}^{M+N} l_{z,(M+k)} p_{y,z})}{\sum_{j=1}^{D} \mathbf{sgn}(af_{i,j})}}{\sum_{w=1}^{A}\sum_{q=1}^{N} ar_{q,w}} = \sum_{i=1}^{A} \frac{\sum_{k=1}^{N}\sum_{y=1}^{D} ar_{k,i}\mathbf{sgn}(af_{i,y})(\sum_{z=1}^{M+N} l_{z,(M+k)} p_{y,z})}{\sum_{j=1}^{D}\mathbf{sgn}(af_{i,j})\sum_{w=1}^{A}\sum_{q=1}^{N} ar_{q,w}} \tag{4}$$

The denominator of Eq. (4) is a scaling factor depending on the application running frequency in FCUs and the numerator is total latency applications encounter while gathering data. To give the details of the numerator, "$\mathbf{L}\times\mathbf{P}$" denotes the latency encountered for reaching data types where they reside, and it is multiplied by the sign function of the data access frequency of the application which is "$\mathbf{sgn}(\mathbf{AF})$". The output of this multiplication gives the latency for an application to access all required data types, and

Table 1
Notations of problem formulation.

Symbol	Definition
Indices	
M	Set of cloud computing data centers
N	Set of fog computing units
D	Set of data types
A	Set of applications
Parameters	
l_{ij}	Latency from data center $i \in M \cup N$ to data center $j \in M \cup N$
uc_i	Used storage capacity of FCU $i \in N$
sc_i	Total storage capacity of FCU $i \in N$
$gv_{j,i}$	Data generation volume for data type $i \in D$ in FCU $j \in N$
dg_i	Total data generation volume for data type i
ar_{ij}	Running frequency of application $j \in A$ in FCU $i \in N$
af_{ij}	Access frequency of application $i \in A$ to data type $j \in D$
Decision variable	
p_{ij}	1 if data type $i \in D$ is placed in data center $j \in M \cup N$, 0 otherwise

finally it is multiplied by the application's running frequency "**AR**" in an FCU which is obtained from the output of DP agent.

sgn(.) is the sign function with the definition as:

$$\mathbf{sgn}(x) = \begin{cases} 1, & if \quad x \in \boldsymbol{R}^{+} \\ 0, & if \quad x = 0 \\ -1, & if \quad x \in \boldsymbol{R}^{-} \end{cases}$$

The reason of using sign function in the formulation is caching. As mentioned in Section 3 for considered time-interval, applications need a small amount of data and when they gather needed data type, they cache it inside FCU.

We can reduce Eq. (4) into a matrix by form using the matrices ***AR***, ***AF***, ***L*** and ***P***. Then we have the following equations in matrix form:

$$\boldsymbol{DL} = \boldsymbol{P} \times \boldsymbol{L} \tag{5}$$

$$= \left[\overrightarrow{dl_1} \quad \overrightarrow{dl_2} \quad \ldots \quad \overrightarrow{dl_{M+N}}\right]$$

$$\boldsymbol{FL} = \left[\overrightarrow{dl_{M+1}} \quad \overrightarrow{dl_{M+2}} \quad \ldots \quad \overrightarrow{dl_{M+N}}\right] \tag{6}$$

$$\boldsymbol{DD} = \mathbf{sgn}(\boldsymbol{AF}) \tag{7}$$

$$= \begin{bmatrix} \overrightarrow{dd_1} \\ \overrightarrow{dd_2} \\ \vdots \\ \overrightarrow{dd_A} \end{bmatrix}$$

$$\boldsymbol{AL} = \boldsymbol{DD} \times \boldsymbol{FL} \tag{8}$$

$$\boldsymbol{ARL} = \boldsymbol{AL} \circ \boldsymbol{AR}^{T} \tag{9}$$

$$\boldsymbol{SF} = \left[\overrightarrow{sf_1} \quad \overrightarrow{sf_2} \quad \ldots \quad \overrightarrow{sf_N}\right] \tag{10}$$

$$\overrightarrow{\boldsymbol{sf_i}} = \|\boldsymbol{AR}\|_{1,1} \times \begin{bmatrix} \left\|\overrightarrow{dd_1}\right\|_1 \\ \left\|\overrightarrow{dd_2}\right\|_1 \\ \vdots \\ \left\|\overrightarrow{dd_2}\right\|_A \end{bmatrix} \tag{11}$$

$$\boldsymbol{AAL} = \boldsymbol{SF} \circ \boldsymbol{ARL} \tag{12}$$

$$\text{Average Latency} = \|\boldsymbol{AAL}\|_{1,1} \tag{13}$$

In Eq. (5), ***DL*** is a $(\boldsymbol{D} \times (\boldsymbol{M} + \boldsymbol{N}))$ matrix and denotes the latencies of obtaining each data (type) from where it is stored. Matrix ***DL*** can be denoted as a row vector of $(\boldsymbol{N} \times 1)$ column vectors and each of these column vectors is displayed as $\overrightarrow{\boldsymbol{dl_i}}$. If we choose the last ***N*** column vectors to form another matrix ***FL*** whose size is $(\boldsymbol{D} \times \boldsymbol{N})$, we can obtain the latency of each FCU for reaching each data type (Eq. (6)). From the applications point of view, data dependencies are important and it is shown by ***DD*** matrix, whose size is $(\boldsymbol{A} \times \boldsymbol{D})$. It is a binary matrix and indicates that if an application has a dependency on the following data or not. We can also show this as a column vector of row vectors, $\overrightarrow{\boldsymbol{dd_i}}$, and each of these indicate the dependency of an application ***i*** on the data types. When the matrices obtained in Eq. (6) and Eq. (7) are multiplied (Eq. (8)), we obtain the latency matrix ***AL*** expressing the latencies that applications encounter while reaching the needed data. It is an $(\boldsymbol{A} \times \boldsymbol{N})$ matrix.

Until now, application running frequencies in FCUs are not taken into account. In order to consider the effect of these profiled running frequencies on latencies, we obtain the ***ARL*** matrix by using the Hadamard product operator ($\circ$) in Eq. (9). This operation is an element-wise product of the entries in matrices ***AL*** and $\boldsymbol{AR}^{T}$, and ***ARL*** is the weighted sum of the latencies applications encounter in which FCU they run. For normalizing the weighted latencies obtained by ***ARL*** matrix, we define a scaling factor ***SF*** in Eq. (10) and Eq. (11). In Eq. (11), $\|.\|_1$ denotes L_1 norm of a vector and $\|.\|_{1,1}$ denotes $L_{1,1}$ norm of a matrix. $L_{p,q}$ norm of an $(m \times n)$ matrix ***A*** is:

$$\|A\|_{p,q} = \left[\sum_{j=1}^{n}\left(\sum_{i=1}^{m}|a_{ij}|^{p}\right)^{(p/q)}\right]^{(1/p)} \tag{14}$$

If the scaling factor and weighted sum latency matrices are multiplied element-wise, we obtain the ***AAL*** matrix, whose size is $(\boldsymbol{A} \times \boldsymbol{N})$ (Eq. (12)). It denotes the average weighted latency of each application running on each FCU. As described in Section 3 the aim is to minimize average latency that applications encounter. Therefore, if the sum of each element in matrix ***AAL*** denoted as $\|.\|_{1,1}$ is minimized, then the goal is achieved. Eq. (13) is the matrix form of Eq. (4).

Eqs. (15) to (18) gives the integer programming model of the problem.

Minimize:

$$\sum_{i \in A} \frac{\sum_{k \in N}\sum_{y \in D} ar_{k,i}\,\mathbf{sgn}(af_{i,y})\left(\sum_{z \in M \cup N} l_{z,(M+k)}\,p_{y,z}\right)}{\sum_{j \in D}\mathbf{sgn}(af_{i,j})\sum_{w \in A}\sum_{q \in N} ar_{q,w}} \tag{15}$$

Subject to:

$$p_{i,j} \in \{0, 1\} \tag{16}$$

$$\sum_{j \in M \cup N} p_{i,j} = 1, \forall i \in D \tag{17}$$

$$\sum_{i \in D} dg_i \times p_{i,(M+j)} \leq sc_j, \forall j \in N \tag{18}$$

In integer programming model, Eq. (15) is the objective function, which is the same as the cost function defined in Eq. (4). Eq. (16) ensures that partial data placement and replication are not allowed, while Eq. (17) guarantees every data type is placed. Finally, Eq. (18) indicates that the storage capacities of the FCUs are not exceeded.

4.2. Data usage

We envision that multiple applications may need the same type of data. Hence, our approach to efficiently store, access and process IoT data involves considering different types of data, like traffic data, health data, etc, separately. Therefore, our approach considers the data in types and applications depend on these data types. In the emergency application in Section 3.2, the data is classified as health and traffic data, and two running applications, navigation and health services, depend on either one or both of them: for navigation services, only traffic data is required, but for health services, both data types are required. Hence, multiple applications may require the same type of data, and an application may require several different types of data. Application-centric data storage approach stores the data for each application separately, but data-centric approach that our architecture is using enables the sharing of the data by multiple applications needing that. The question is how efficient the proposed architecture is instead of using application-centric data storage. Before answering this question, we need to formulate the relationship between applications and their data requirements.

The matrix **AF**, defined in Section 4.1, indicates which applications need which data types. But this is not a compact form to express the sharing amount of data among the applications (i.e., data overlap ratio), that means in how many applications require a specific data type.

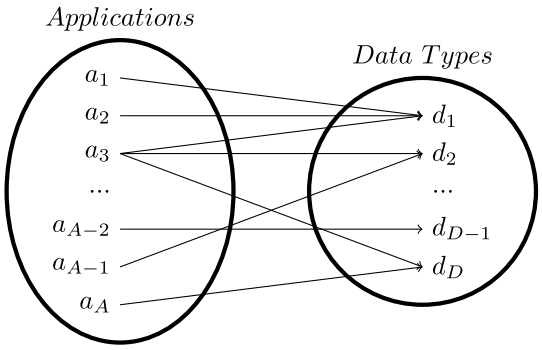

Fig. 3. Example application — data type relation graph.

We can use a simple bipartite graph as well to show the data needs of applications. The vertices are partitioned into two sets: applications and data types. The edges of the bipartite graph denote the dependencies of applications on data types, which are also same as the non-zero elements of **AF** matrix. An example of this bipartite graph is given in Fig. 3.

If the given example graph is considered, we can easily say that both applications a_1 and a_2 require data type d_1, a_3 requires d_1, d_2 and d_D and so on. It is obvious in such a case that some of the data types are required by multiple applications, and if application-centric data storage would be used, the data would be replicated which would lead to unnecessary storage costs.

In our model, whenever data of some type is generated it has to be required by one or more applications to be stored. Unused data types are not stored. Hence, we assume that all data types are used. This means each data type is used at least once. This gives the baseline definition of data excess usage (sharing amount or overlap ratio) and is defined in Eq. (19).

$$\textbf{ExcessUse} = \frac{\sum_{i=1}^{A}\sum_{j=1}^{D}\textbf{sgn}\left(af_{ij}\right)}{D} \tag{19}$$

Eq. (19) gives a general idea about how data types set is enclosed. It defines whether a data type is used by more than one application or not.

$$\textbf{ExcessUse} = 1 \tag{20}$$

If Eq. (20) is satisfied in the network then all data types are used by different applications, there is no data type which has been used by multiple applications.

$$\textbf{ExcessUse} > 1 \tag{21}$$

If Eq. (21) is the case, then there is at least one data type used by multiple applications.

Relation between AF and excess usage

The value of excess usage depends on **AF** matrix (Eq. (19)) whose size is defined by **A** and **D**. Therefore, the values of excess usage is related with both **A** and **D** parameters, and there are three conditions:

1. $\textbf{A} > \textbf{D}$: This is the most probable case, since lots of applications can be developed by using existing data types.

Lemma 4.1. *If this is the case then:*

$$\textbf{ExcessUse} > 1$$

and values of excessive usage is limited with $[A/D, A]$.

Proof. At least one data type is required for an application to run correctly, this is the condition and obligation to data types in the network. If data and applications are assumed as two sets, each data type and application become a member of these sets. By using the pigeonhole principle, at least $A - D$ applications are left, after one-to-one mapping of these sets are done. The number of non-zero elements in **AF** becomes **A** and the following equation is satisfied:

$$\textbf{ExcessUse} = \textbf{A}/\textbf{D}$$

$$\textbf{A} > \textbf{D} \rightarrow \textbf{ExcessUse} > 1.$$

In the most extreme case, all applications may use all data types. In that case then:

$$\textbf{ExcessUse} = \textbf{A}. \quad \square$$

2. $\textbf{A} = \textbf{D}$: This case is probable in the architecture, since there is no limitation or relation between application count and data count.

Lemma 4.2. *If this is the case then:*

$$\textbf{ExcessUse} \geq 1$$

and values of excessive usage is limited with $[1, A]$.

Proof. Since application and data type counts are equal in this case, if all applications require one data type, which is a valid assumption throughout the architecture, then in one-to-one mapping of application and data type sets cover each other. This means no excess usage occurs, which sets

$$\textbf{ExcessUse} = 1.$$

In the most extreme case, all applications may use all data types. In that case:

$$\textbf{ExcessUse} = \textbf{A}. \quad \square$$

3. $\textbf{A} < \textbf{D}$: This case is the most unlikely among others. In general, application count is greater than data type count, since many more applications can be designed with available data types.

Lemma 4.3. *If this is the case then:*

$$\textbf{ExcessUse} \geq 1$$

and values of excessive usage is limited with $[1, A]$.

Proof. If one-to-one mapping of applications and data types is assumed, then there exists some data types not used by any of the applications. It is a contradiction, since unused data types are not stored. So at least one application requires these uncovered data types. Again by using the pigeonhole principle, all data set is covered by applications, which makes at least **D** non-zero elements in **AF** matrix. So excessive usage satisfies

$$\textbf{ExcessUse} = 1.$$

In the most extreme case, all applications may use all data types. In that case then:

$$\textbf{ExcessUse} = \textbf{A}. \quad \square$$

Given cases are the theoretical limits of excess usage in the architecture and help for setting up our performance experiments.

5. Proposed algorithms

Section 4 reduces the problem in our proposed architecture to an integer programming problem. Then we can use any state of the art commercial solvers such as Gurobi [45] or CPLEX [46] to solve our problem. The solution time for these solvers increase exponentially with the number of data centers, data types and running applications.

To overcome this, we propose two heuristic algorithms for reducing the cost function and obtaining a solution close to the optimal one. In both algorithms, we assume that data generation volumes, data access patterns and running frequencies of the applications are known a priori. As explained in Section 3, these values can be tracked by using DC and DP agents that run in FCs. Since all IoT nodes are the elements of FCUs, all data generation and consumption incidents occur in FCUs. DC and DP agents, together, can monitor all data generation and usage characteristics easily, and provide the necessary information for constructing **GV**, **AR** and **AF** matrices.

Any one of the cloud data centers available in the network can process the proposed algorithms. The only requirement for algorithms to run correctly is to gather the outputs of DC and DP agents, and we can achieve this by using a centralized mechanism, such as software-defined networking (SDN) controller.

5.1. Algorithm 1: placement of mostly accessed data first

Algorithm 1: Placement of mostly accessed data to the nearest data center where it is mostly used.

Data: M, N, D, A, L, AR, AF, GV, SC
Result: P
1 $dataAccess \leftarrow$ **matrixMultiply**(AR, AF)
2 $DG \leftarrow$ **matrixColumnSum**(GV)
3 $tempAccess \leftarrow$ **matrixColumnSum**($dataAccess$)
4 **for** $i \leftarrow 0$ **to** $D-1$ **do**
5 $\quad dataUse[0][i] \leftarrow i$
6 $\quad dataUse[1][i] \leftarrow tempAccess[i] \times DG[i]$
7 **mergeSortDescendingWIndices**(dataUse,0,D-1)
8 $P \leftarrow$ **placeDataTypeAlg1**(M, N, D, L, SC, DG, $dataAccess$, $dataUse$)
9 **return** P

Our first heuristic algorithm, Algorithm 1, places mostly accessed data types to the nearest data centers where they are mostly used without considering application running profiles. DP agent can monitor application run profiles, and data access patterns, while DC agent can give the statistics of data generation volumes. Using the outputs of DC and DP, data types are sorted according to their total access volumes and placement starts from the most accessed data type. Rule is to find for each data type a data center (an FCU) which is nearest to the FCU where it is mostly used. If the remaining storage capacity of the nearest FCU is not big enough to store the data (of some type), the second best FCU for that data type is chosen, and this continues until data is placed. Convergence of the algorithm is guaranteed by the existence of the CCs which have unlimited storage capacities. If the capacity of the nearby FCUs are not big enough, the data is placed to the nearest CC.

In the pseudo-code of the algorithm, from line 1 to 7, data access volume for each data type is calculated. It is an indication of how much each data type is used by all FCUs. After the calculation, they are sorted from the most accessed to the least accessed one. This process takes $\mathbf{O}(N \times A \times D)$ time without using any kind of matrix multiplication optimization. Runtime of the Algorithm 1 is dominated by data access calculation which requires the multiplication of matrices. Sorting and calculating data access volumes do not take that much of time. The placement procedure of the Algorithm 1 takes $\mathbf{O}(D \times (M+N))$. In the placement procedure, from line 1 to line 2 the initial values of used capacities and placement data matrix are set. Starting from line 3, the most accessed data type is placed to the nearest data center available where it is mostly used. The overall runtime of the algorithm is $\mathbf{O}(N \times A \times D)$ because the total number of CCs is much less than the number of FCUs, and at least one application runs in the system. For achieving better performance, the bottleneck of the algorithm should be improved, which is a rectangular matrix–matrix multiplication.

Procedure placeDataTypeAlg1(M, N, D, L, SC, DG, dataAccess, dataUse)

1 Initialize P matrix: $P \leftarrow 0$
2 Initialize UC vector: $UC \leftarrow 0$
3 **for** $i \leftarrow 0$ **to** $D-1$ **do**
4 $\quad tempMax \leftarrow 0$
5 $\quad maxInd \leftarrow 0$
6 $\quad$ **for** $j \leftarrow 0$ **to** $N-1$ **do**
7 $\quad\quad$ **if** $dataAccess[j][dataUse[0][i]] > tempMax$ **then**
8 $\quad\quad\quad tempMax \leftarrow dataAccess[j][dataUse[0][i]]$
9 $\quad\quad\quad maxInd \leftarrow j$
10 $\quad minDist \leftarrow \infty$
11 $\quad minInd \leftarrow 0$
12 $\quad$ **for** $j \leftarrow 0$ **to** $M+N-1$ **do**
13 $\quad\quad$ **if** $L[M+maxInd][j] < minDist$ **then**
14 $\quad\quad\quad$ **if** $j < M$ **then**
15 $\quad\quad\quad\quad minDist \leftarrow L[M+maxInd][j]$
16 $\quad\quad\quad\quad minInd \leftarrow j$
17 $\quad\quad\quad$ **else if** $UC[j-M] + DG[i] \leq SC[j-M]$ **then**
18 $\quad\quad\quad\quad minDist \leftarrow L[M+maxInd][j]$
19 $\quad\quad\quad\quad minInd \leftarrow j$
20 $\quad$ **if** $minInd \geq M$ **then**
21 $\quad\quad UC[minInd-M] \leftarrow UC[minInd-M] + DG[i]$
22 $\quad P[i][minInd] \leftarrow 1$
23 **return** P

5.2. Algorithm 2: Hybrid placement considering data access patterns & applications running profiles

Algorithm 2: Placement of data that affect cost function most to the nearest data center where it is used most according to best choices.

Data: M, N, D, A, L, AR, AF, GV
Result: P
1 $DG \leftarrow$ **matrixColumnSum**(GV)
2 **mergeSortDescendingWIndices**(DG,0,D-1)
3 $normDenom \leftarrow \sum_{i=0}^{N-1} \sum_{j=0}^{A-1} ar_{i,j}$
4 $NAR \leftarrow AR/normDenom$
5 $rowScaling \leftarrow$ **matrixRowSum**(**sgn**(AF))
6 **for** $i \leftarrow 0$ **to** $A-1$ **do**
7 $\quad$ **for** $j \leftarrow 0$ **to** $D-1$ **do**
8 $\quad\quad SFD[i][j] \leftarrow$ **sgn**$(AF[i][j])/rowScaling[i]$
9 $DCF \leftarrow$ **matrixMultiply**(NAR, SFD)
10 $P \leftarrow$ **placeDataTypeAlg2**(M, N, D, L, SC, DG, DCF)
11 **return** P

The idea behind in Algorithm 2 is to find a suitable place for data types starting from the most effective ones on defined average latency function. In the algorithm, efficiencies of each data type are calculated according to variables defined in Section 4, and then placement choices for each data type are ordered for minimizing the average latency. Since the problem is a linear optimization

Procedure placeDataTypeAlg2(M, N, D, L, SC, DG, DCF)

```
1  Initialize data placed vector: dataPlaced ← 0
2  Initialize P matrix: P ← 0
3  Initialize UC vector: UC ← 0
4  placedDataCnt ← 0
5  turn ← 0
6  while placedDataCnt ≠ D do
7      for i ← 0 to D do
8          if dataPlaced[DG[0][i]] ≠ 1 then
9              for j ← 0 to M + N − 1 do
10                 pL[0][j] ← 0
11                 pL[1][j] ← 0
12                 for k ← 0 to M + N − 1 do
13                     if k ≥ M then
14                         pL[1][j] ←
                           pL[1][j] + (L[j][k] × DCF[k − M][DG[0][i]])
15             mergeSortAscendingWIndices(pL,0,M+N-1)
16         if pL[0][turn] ≥ M then
17             if UC[pL[0][turn] − M] + DG[1][i] ≤ SC[pL[0][turn] − M]
               then
18                 P[DG[0][i]][pL[0][turn]] ← 1
19                 UC[pL[0][turn] − M] ←
                   UC[pL[0][turn] − M] + DG[1][i]
20                 dataPlaced[DG[0][i]] ← 1
21                 placedDataCnt ← placedDataCnt + 1
22         else
23             P[DG[0][i]][pL[0][turn]] ← 1
24             dataPlaced[DG[0][i]] ← 1
25             placedDataCnt ← placedDataCnt + 1
26     turn ← turn + 1
27 return P
```

procedure, each decision made affects the whole placement. Regarding that, placing each data type to their best choice is not usually possible, so it is thought that placing in turn will converge to the optimum solution. For each round, remaining choices are sorted, and data is placed if the capacity of the best chosen data center is available. This seems complicated, but can become clear by the following example. Consider three data types to be placed in two FCUs and a CC. Starting from most effective data type, for example data type 1, FCU 1 minimizes the average latency. Then, data type 1 is placed to FCU 1, after that second most effective data type is chosen, which may be data type 2. Let its first choice be again FCU 1, but the remaining capacity of FCU 1 is not available for storing another big data, so it is left unplaced. Now comes to the last data type, which is data type 3, and let CC be its first choice that minimizes the latency, then it is placed in CC. No other data types are left except data type 2, and the first placement turn is over since all best choices for each data type are visited. After the first turn, the second round is started from the most effective remaining data type on the average latency that has not been placed yet. In the second round, next best choices are taken into account. Now, assume that second best place for data type 2 is FCU 2, but again its capacity is not big enough for storing the data type 2, then third best choice comes into play in the next round. For this example, it must be CC, since no other data centers are left and every data has to be placed in one data center, then the final place for data type 2 is CC. This algorithm can be assumed as a tokenized algorithm.

Algorithm 2 considers the normalized application running frequencies together with data requirement pattern. This distinguishes Algorithm 2 from Algorithm 1. Data requirement pattern means, whenever an application accesses a certain data type, it is marked as needed by the application. Therefore, applications' data access frequency matrix denoted by **AF** is changed to a binary matrix: if an application's dependency on a data type is greater than '0', then it is changed to '1', this is also the case in formal problem formulation in Eq. (4), otherwise it is left as '0'.

The algorithm starts from calculating which data types are generated mostly in the given network, from line 1 to 2, and it takes $\mathbf{O}(D \times N)$ according to the assumption that sorting takes less time than calculating data generation volumes. From line 3 to 8 normalized application running frequencies and data dependencies are calculated. Calculation of normalized application running frequencies takes $\mathbf{O}(N \times A)$ and changing **AF** matrix to binary takes $\mathbf{O}(A \times D)$ time. After obtaining normalized application running frequencies and data access matrices, they are multiplied with each other for calculating which data types are required in which FCU and how often. If no optimized matrix multiplication algorithm is used, it would take $\mathbf{O}(N \times A \times D)$. Then placement starts, from line 1 to 3 in the placement procedure, initializations of used capacity and data placement variables are done. It takes $\mathbf{O}(D \times (M + N))$, which is same for both algorithms. In data placement variables, there exists a flag indicating whether the corresponding data type is placed or not, and this is used for terminating the placement algorithm with two other variables: **placedDataCnt** and **turn**. One of them is used for checking whether all data types are placed or not, and the second one indicates in which round the placement algorithm is.

Starting from line 6 of the procedure, Algorithm 2 places each data type to their best choices turn-by-turn. Runtime of the Algorithm 2 is $\mathbf{O}(D \times (M + N)^2)$.

The major advantage of Algorithm 1 and Algorithm 2 is that they can easily be parallelized, since they consist of mostly matrix multiplications. Parallelization potential of the proposed algorithms makes them suitable for running in multi-core architectures efficiently.

6. Experimental results

6.1. Experimental setup

For testing the performance of the proposed algorithms in our simulations, we define a fixed rectangular area to present a large geographical region and then divide the region into smaller rectangles. We assume that each small rectangle is an FCU (fog computing unit) and edges of the FCUs are the possible candidates for placing CCs (cloud data centers). This topology resembles a random smart city network, so that every neighborhood in the city is assumed as an FCU, consists of a fog data center and its connected IoT nodes. The intersections of the neighborhoods are possible candidate locations for CCs. Places of CCs are chosen according to hierarchical merging, which we express as follows. Every FCU is modeled as a point where all data is either generated or consumed. The nearest two FCU points form a cluster and formed clusters grow until no FCU points are left. When algorithm finishes, the closest candidate CC point to the center of the cluster is chosen for placing cloud computing data center. An example run of our topology generation algorithm is given in Fig. 4. We can use this algorithm also inside FCUs for establishing the connections between IoT nodes and fog computing data centers. We can assume each rectangular area as IoT nodes in a region and fog computing data centers as cloud computing ones explained in the previous example.

For generating **AR** matrix, a metric is defined for describing at least how many applications run in an FCU, and it is called application run ratio (**appUseRatioInFCU**). Its formula is given in Eq. (22).

$$\mathbf{appUseRatioInFCU_i} = \frac{\sum_{j=1}^{A} \mathbf{sgn}(ar_{ij})}{A} \tag{22}$$

For a given **appUseRatioInFCU**, **AR** matrix is generated satisfying that at least one application runs in an FCU, and each application runs in at least one FCU.

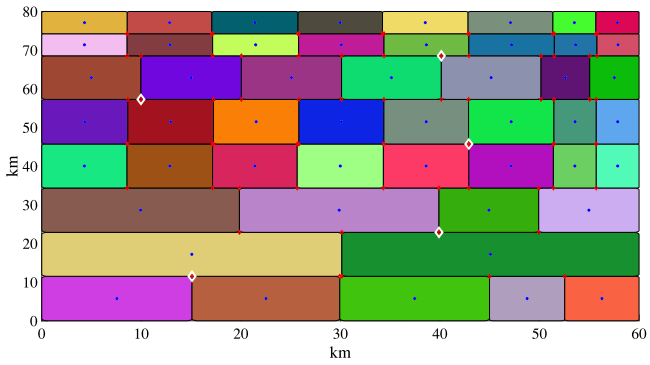

Fig. 4. Smart City Example Network: M = 5, N = 50. Blue points are FCU centers and red ones are possible CC locations. After using hierarchical merging chosen CC points are marked with white. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Experiments are made with MATLAB [47] and linear solver Gurobi 7.5.2 [45], which is called inside MATLAB by using its API. A fixed 60km by 80km rectangular area is defined for all simulations, and delay from one center to the another is calculated by dividing two times the distance by the speed of light. We assume that connections between FCUs and CCs are established with high capacity either fiber optic or wireless links. For each sweeping variable, we run 100 samples of simulations and report the averages of these 100 runs as our results. For all scenarios investigated, application count (**A**), data type count (**D**), total data generation volume of one data type for a fixed period, maximum value of AR and AF are same, and 45, 20, ∼N(1 GB, 256 MB), ∼U[0,3] and ∼U[0,3] respectively.

6.2. Comparison of the algorithms

We compare our proposed algorithms with both random placement heuristic, data-centric and application-centric solutions in aforementioned rectangular network topology. Abbreviations of the algorithms are given in Table 2.

Table 2
Algorithms used in experiments.

Alg1	First proposed algorithm
Alg2	Second proposed algorithm
RP	Data is placed randomly
LPDC	Optimal linear solution for data-centric placement
LPAC	Optimal linear solution for application-centric placement

Alg1 is the first proposed algorithm (Algorithm 1) explained in Section 5.1.

Alg2 is the second proposed algorithm (Algorithm 2) explained in Section 5.2.

RP is the random placement algorithm. All data types are randomly placed in FCUs and CCs while taking into account the storage capacity limitations.

LPDC denotes the output of linear solver for the optimization model from Eqs. (15)–(18) in Section 4.1.

LPAC denotes the output of the linear solver for application-centric approach. It is slightly different version of the **LPDC**, since data is not typed, and data access frequency denoted by **AF** matrix in data-centric model is considered in the capacity constraint of the **LPAC** linear model. The corresponding optimization model for **LPAC** is given from Eqs. (23)–(26).

Minimize:

$$\sum_{i\in A} \frac{\sum_{k\in N} ar_{k,i} \sum_{z\in M\cup N} l_{z,(M+k)} p_{i,z}}{\sum_{w\in A}\sum_{q\in N} ar_{q,w}} \tag{23}$$

Subject to:

$$p_{i,j} \in \{0, 1\} \tag{24}$$

$$\sum_{j\in M\cup N} p_{i,j} = 1, \forall i \in A \tag{25}$$

$$\sum_{i\in A} adr_i \times p_{i,(M+j)} \leq sc_j, \forall j \in N \tag{26}$$

All variables used in Eqs. (23)–(26) are same as the variables used in **LPDC** except with one minor difference and a new variable. As the minor difference, **LPAC** placement variable, $\mathbf{p_{i,j}}$, denotes the required data for application **i** is placed in data center **j** which is an FCU or a CC, where it denotes the final place of data type **i** in **LPDC**. A new variable **ADR** is introduced for denoting the total size of the data required by the applications which is a vector of size ($\boldsymbol{A} \times 1$) (Eq. (27)). It is obtained by multiplication of matrix **AF** with vector **DG** whose size is ($\boldsymbol{D} \times 1$) and defined in Eq. (1).

$$\mathbf{ADR} = \mathbf{AF} \times \mathbf{DG} \tag{27}$$

6.3. Analysis of the results

In the first experimental setup, we investigate a sample network where storage capacities of FCUs are normally distributed with mean 7 GB and variance 1 GB. Application run ratio and excess use parameters are set to 0.1 and 30, respectively. In the simulated network, data dependencies of applications are relatively high, but in each FCU, only 5 out of 45 different applications run. Average latency encountered by applications is given in Fig. 5, required bandwidth assuming all generated data is accessed by applications in a second is shown in Fig. 6, and finally storage capacities required for the simulated network is available in Fig. 7.

If we consider latency perspective (Fig. 5), average latencies of all algorithms we use in experiments increase as FCU count reaches to 150. Average latency of Alg1 is slightly better than RP, but both of them are worse than the others. Increase in the latency of application-centric approach is higher than data-centric, and when FCU count is greater than 123, it becomes worse than both Alg2 and data-centric approach. Alg2 is worse than the optimum, but it tracks optimal solution in '0.1%' on the average. From network occupancy perspective, data-centric optimal solution is '1%' less than application-centric and proposed Algorithm 2 is same as data-centric optimum. As expected, huge amount of storage (937.5 GB) is required for application-centric data placement approach where small amount of storage (20.0 GB) is enough for data-centric placement.

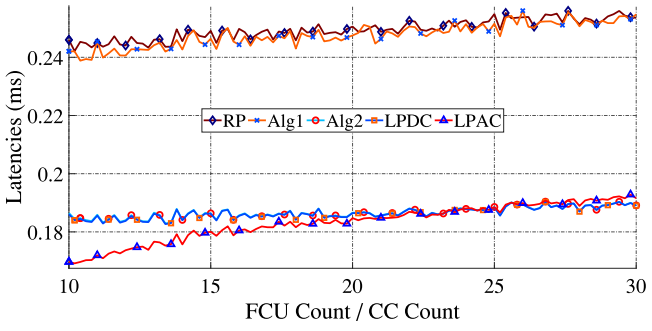

Fig. 5. Average latency vs N/M. where M = 5 & storage capacity available in each FCU ∼ N(7 GB, 1 GB).

Second setup is used for understanding the effect of how average latency changes while applications running count increases

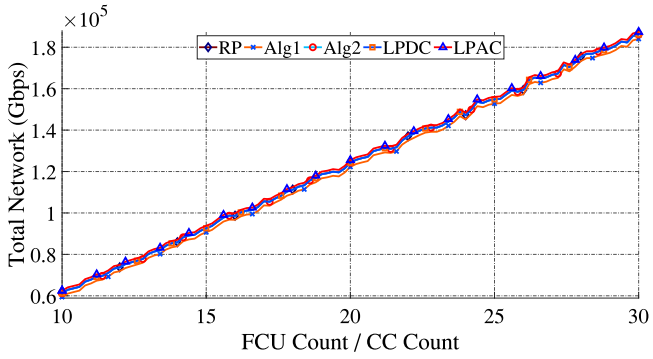

Fig. 6. Network occupancy vs N/M. where M = 5 & storage capacity available in each FCU ∼ N(7 GB, 1 GB).

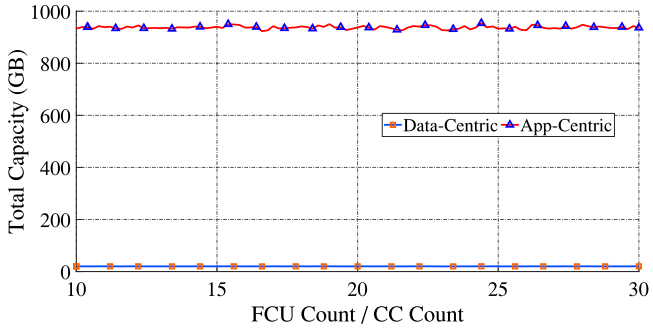

Fig. 7. Total storage capacity vs N/M. where M = 5 & storage capacity available in each FCU ∼ N(7 GB, 1 GB).

on each FCU. As the number of IoT nodes in an FCU increases, the probability of running more applications in FCUs also increases. So we fix the excess use parameter to 10 and sweep application run ratio from '0.2' to '0.75' with an increase of FCU count from 50 to 150. Data dependencies are less than the first setup, but more applications run in each FCU. In the worst case 9 out of 45 different applications run in each FCU, and at the final step 34 out of 45 applications run. We present the average latencies of the mentioned scenarios in Fig. 8. When more applications run in each FCU, data-centric approach overwhelms application-centric one, and proposed Algorithm 2 tracks the optimal solution.

As a final scenario, we consider the scenario when FCU count increases drastically in which all applications run in each of the FCUs. So the number of FCUs is swept from 50 to 500, and 45 out of 45 applications run in each of these FCUs (see Fig. 9). This scenario can be possible when Social Internet of Things (SIoT) concept is deployed. The interesting thing in this scenario is at some point latencies start to decrease. This can be expected as there is a limited rectangular area which is 60km by 80km, and it is divided into many more small rectangles such as 500 smaller ones. Therefore, center distances of these rectangles decrease, and also the average latency. Also in this case, due to the reason mentioned in the second scenario, data-centric approach performs better than application-centric placement by about '10%'.

All simulations verify that Algorithm 2 proposed in this paper gets very close to the optimal solution. For networks composed of FCUs and CCs, if running frequencies of applications on FCUs are high, data-centric placement approach outperforms the application-centric one, and it also drastically decreases storage costs required on the network.

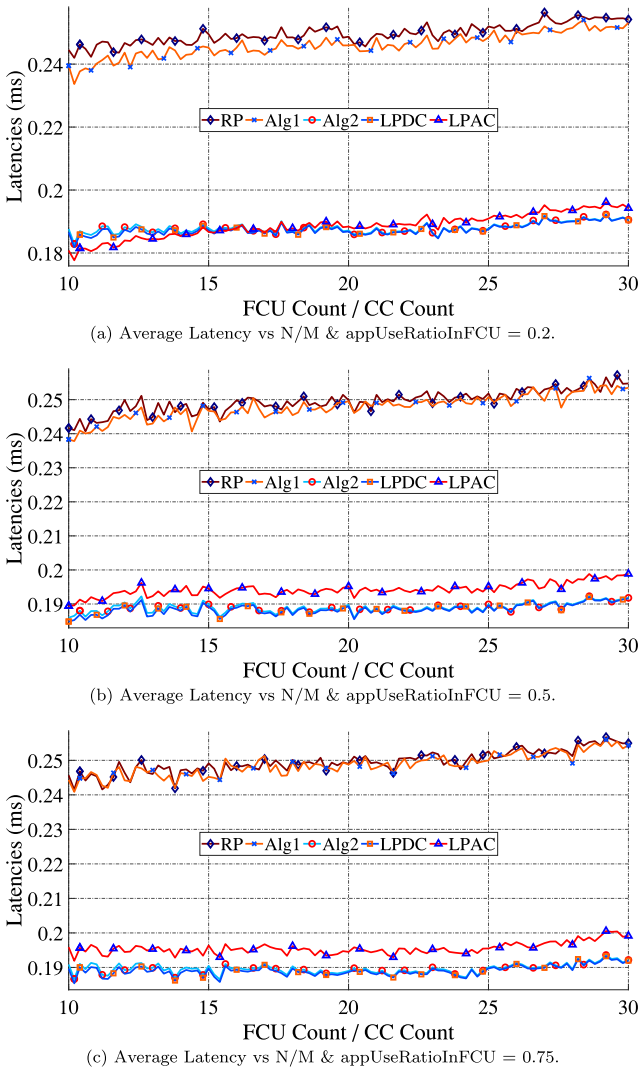

Fig. 8. Effect of application run ratio on average latency M = 5 & excessuse = 10.

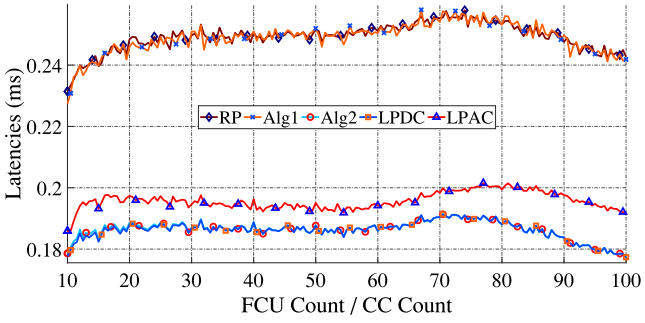

Fig. 9. Average Latency vs N/M. where M = 5 & excessuse = 10 & appUseRatioInFCU = 1.0.

7. Conclusion

In this paper, we first propose a cloud computing (CC) and fog computing (FC) based IoT architecture to efficiently place and serve IoT data. We consider the fact that the same type of IoT data may be needed and used by multiple applications. Therefore, we propose the classification of IoT data into types and identification of which applications may require which type of data. We model the data

placement problem as an optimization problem where latency encountered by applications in accessing data is to be minimized.

We also propose data placement strategies and algorithms for reducing the average latency encountered by applications. Proposed data placement strategies depend on the classification of data into types for reducing the network storage costs using Fog Computing Units (FCUs). Data Classifier (DC) and Data Profiler (DP) agents run in fog computing data centers are proposed to be responsible for classification of data and monitoring applications' run and data access profiles. Data is distributed among FCU and CC resources without replicating for each application separately, which we called data-centric placement. This significantly reduces storage costs required for handling big data generated in IoT networks. We propose two algorithms for data placement, and we conducted extensive simulation experiments to evaluate them. Our simulation results show that our algorithms can efficiently place data without increasing average latency that applications encounter. We also observed that when application running frequencies on FCUs increases, data-centric placement performs much better than application-centric data placement where each application stores the needed data separately and independently.

Since we mainly focus on the hierarchical architecture and placement algorithms together with their analyses, we leave the comparison of our proposed algorithms (Algorithm 1 and Algorithm 2) with meta-heuristic algorithms, such as genetic algorithms, simulated annealing, particle swarm optimization (PSO), as future work. We also leave the details of establishing connections between fog computing data centers and IoT nodes out of the scope of this work. For different metrics, various connection mechanisms can be studied as new research issues. Although we do not consider replication and partial placement in this paper, they may be required for some cases, for example, replication comes into play when reliability is critical. These issues are left as potential future work of our proposed model.

Our work is complementary with some other approaches and methods that try to optimize different objectives, and focus on a different aspect of the resource allocation problem. There are works, for example, that focus on application placement or virtual machine placement considering available server and network resources. One can integrate our work with such resource allocation algorithms to have a more comprehensive placement and resource allocation solution.

We believe that this work will guide fog based IoT network designers in designing efficient network architectures and data placement strategies, which will become more important as the number of IoT nodes and data-intensive IoT applications proliferates.

Acknowledgment

This work is supported in part by The Scientific and Technological Research Council of Turkey (TUBITAK) with Grant 116E048.

References

[1] ITU Internet Reports The Internet of Things, Tech. Rep., ITU, Geneva, Nov. 2005.
[2] D. Evans, The internet of things how the next evolution of the internet is changing everything, Apr. 2011, Available at https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. (Accessed 28 July 2018).
[3] L. Atzori, A. Iera, G. Morabito, The Internet of Things: A survey, Comput. Netw. 54 (15) (2010) 2787–2805, http://dx.doi.org/10.1016/j.comnet.2010.05.010.
[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of Things: A survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutor. 17 (4) (2015) 2347–2376, http://dx.doi.org/10.1109/COMST.2015.2444095.
[5] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645–1660, http://dx.doi.org/10.1016/j.future.2013.01.010.
[6] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I.S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer, P. Doody, Internet of Things strategic research roadmap, in: O. Vermesan, P. Friess (Eds.), Internet of Things - Global Technological and Societal Trends From Smart Environments and Spaces to Green ICT, River Publishers, Aalborg, 2011, pp. 9–52 (Chapter 2).
[7] R.K. Ganti, F. Ye, H. Lei, Mobile crowdsensing: current state and future challenges, IEEE Commun. Mag. 49 (11) (2011) 32–39, http://dx.doi.org/10.1109/MCOM.2011.6069707.
[8] P. Scully, The Top 10 IoT segments in 2018 - based on 1600 real IoT projects, Feb. 2018, Available at http://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/. (Accessed 28 July 2018).
[9] A. Zanella, N. Bui, A. Castellani, L. Vangelista, M. Zorzi, Internet of Things for smart cities, IEEE Internet Things J. 1 (1) (2014) 22–32, http://dx.doi.org/10.1109/JIOT.2014.2306328.
[10] H. Schaffers, N. Komninos, M. Pallot, B. Trousse, M. Nilsson, A. Oliveira, Smart cities and the future internet: Towards cooperation frameworks for open innovation, in: The Future Internet, Springer, Berlin, 2011, pp. 431–446, http://dx.doi.org/10.1007/978-3-642-20898-0_31.
[11] L.D. Xu, W. He, S. Li, Internet of Things in industries: A survey, IEEE Trans. Ind. Inf. 10 (4) (2014) 2233–2243, http://dx.doi.org/10.1109/TII.2014.2300753.
[12] M. Chen, S. Mao, Y. Liu, Big data: A survey, Mobile Netw. Appl. 19 (2) (2014) 171–209, http://dx.doi.org/10.1007/s11036-013-0489-0.
[13] F. Bonomi, R. Milito, P. Natarajan, J. Zhu, Fog computing: A platform for internet of things and analytics, in: N. Bessis, C. Dobre (Eds.), Big Data and Internet of Things: A Roadmap for Smart Environments, Springer, Cham, 2014, pp. 169–186, http://dx.doi.org/10.1007/978-3-319-05029-4_7.
[14] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, M. Zaharia, Above The Clouds: A Berkeley View of Cloud Computing, Tech. Rep., University of California at Berkeley, 2009.
[15] I.A.T. Hashem, I. Yaqoob, N.B. Anuar, S. Mokhtar, A. Gani, S.U. Khan, The rise of "big data" on cloud computing: Review and open research issues, Inf. Syst. 47 (2015) 98–115, http://dx.doi.org/10.1016/j.is.2014.07.006.
[16] A. Botta, W. de Donato, V. Persico, A. Pescapé, Integration of cloud computing and internet of things: A survey, Future Gener. Comput. Syst. 56 (2016) 684–700, http://dx.doi.org/10.1016/j.future.2015.09.021.
[17] F. Bonomi, R. Milito, J. Zhu, 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, MCC '12, ACM, New York, NY, USA, 2012, pp. 13–16, http://dx.doi.org/10.1145/2342509.2342513.
[18] S. Yi, C. Li, Q. Li, A survey of fog computing: concepts, applications and issues, in: Proceedings of the 2015 Workshop on Mobile Big Data, Mobidata '15, ACM, New York, NY, USA, 2015, pp. 37–42, http://dx.doi.org/10.1145/2757384.2757397.
[19] I. Stojmenovic, S. Wen, The fog computing paradigm: Scenarios and security issues, in: 2014 Federated Conference on Computer Science and Information Systems, vol. 2, 2014, pp.1–8, http://dx.doi.org/10.15439/2014F503.
[20] L.M. Vaquero, L. Rodero-Merino, Finding your way in the fog: towards a comprehensive definition of fog computing, SIGCOMM Comput. Commun. Rev. 44 (5) (2014) 27–32, http://dx.doi.org/10.1145/2677046.2677052.
[21] J. Deichmann, K. Heineke, T. Reinbacher, D. Wee, Creating a successful Internet of Things data marketplace, Oct. 2016, Available at http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/creating-a-successful-internet-of-things-data-marketplace. (Accessed 28 July 2018).
[22] J. Santos, T. Wauters, B. Volckaert, F. De Turck, Fog computing: Enabling the management and orchestration of smart city applications in 5G networks, Entropy 20 (1) (2018) 4, 1–26, http://dx.doi.org/10.3390/e20010004.
[23] M. Aazam, E.N. Huh, Fog computing and smart gateway based communication for cloud of things, in: 2014 International Conference on Future Internet of Things and Cloud, 2014, pp. 464–470, http://dx.doi.org/10.1109/FiCloud.2014.83.
[24] M. Jutila, An adaptive edge router enabling Internet of Things, IEEE Internet Things J. 3 (6) (2016) 1061–1069, http://dx.doi.org/10.1109/JIOT.2016.2550561.
[25] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768, http://dx.doi.org/10.1016/j.future.2011.04.017.
[26] Z. Guo, S. Hui, Y. Xu, H.J. Chao, Dynamic flow scheduling for power-efficient data center networks, in: 2016 IEEE/ACM 24th International Symposium on Quality of Service, IWQoS, 2016, pp. 1–10, http://dx.doi.org/10.1109/IWQoS.2016.7590399.
[27] Z. Guo, Z. Duan, Y. Xu, H.J. Chao, Cutting the electricity cost of distributed datacenters through smart workload dispatching, IEEE Commun. Lett. 17 (12) (2013) 2384–2387, http://dx.doi.org/10.1109/LCOMM.2013.102213.131831.
[28] Z. Guo, Z. Duan, Y. Xu, H.J. Chao, JET: Electricity cost-aware dynamic workload management in geographically distributed datacenters, Comput. Commun. 50 (2014) 162–174, http://dx.doi.org/10.1016/j.comcom.2014.02.011.

[29] V. Angelakis, I. Avgouleas, N. Pappas, E. Fitzgerald, D. Yuan, Allocation of heterogeneous resources of an IoT device to flexible services, IEEE Internet Things J. 3 (5) (2016) 691–700, http://dx.doi.org/10.1109/JIOT.2016.2535163.
[30] C.W. Tsai, SEIRA: An effective algorithm for IoT resource allocation problem, Comput. Commun. 119 (2018) 156–166, http://dx.doi.org/10.1016/j.comcom.2017.10.006.
[31] K. Toczé, S. Nadjm-Tehrani, A taxonomy for management and optimization of multiple resources in edge computing, Wirel. Commun. Mob. Comput. 2018 (2018) 7476201, 1–23, http://dx.doi.org/10.1155/2018/7476201.
[32] R. Deng, R. Lu, C. Lai, T.H. Luan, H. Liang, Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption, IEEE Internet Things J. 3 (6) (2016) 1171–1181, http://dx.doi.org/10.1109/JIOT.2016.2565516.
[33] L. Tong, Y. Li, W. Gao, A hierarchical edge cloud architecture for mobile computing, in: IEEE INFOCOM 2016 - IEEE International Conference on Computer Communications, 2016, pp. 1–9, http://dx.doi.org/10.1109/INFOCOM.2016.7524340.
[34] D. Zeng, L. Gu, S. Guo, Z. Cheng, S. Yu, Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system, IEEE Trans. Comput. 65 (12) (2016) 3702–3712, http://dx.doi.org/10.1109/TC.2016.2536019.
[35] H.R. Arkian, A. Diyanat, A. Pourkhalili, MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications, J. Netw. Comput. Appl. 82 (2017) 152–165, http://dx.doi.org/10.1016/j.jnca.2017.01.012.
[36] O. Skarlat, S. Schulte, M. Borkowski, P. Leitner, Resource provisioning for IoT services in the fog, in: 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA, 2016, pp. 32–39, http://dx.doi.org/10.1109/SOCA.2016.10.
[37] R. Yu, G. Xue, X. Zhang, Application provisioning in FOG computing-enabled Internet-of-Things: A network perspective, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, 783–791, http://dx.doi.org/10.1109/INFOCOM.2018.8486269.
[38] Y. Qin, Q.Z. Sheng, N.J. Falkner, S. Dustdar, H. Wang, A.V. Vasilakos, When things matter: A survey on data-centric internet of things, J. Netw. Comput. Appl. 64 (2016) 137–153, http://dx.doi.org/10.1016/j.jnca.2015.12.016.
[39] B. Yu, J. Pan, Location-aware associated data placement for geo-distributed data-intensive applications, in: IEEE INFOCOM 2015 - 2015 IEEE Conference on Computer Communications, 2015, pp. 603–611, http://dx.doi.org/10.1109/INFOCOM.2015.7218428.
[40] B. Tang, Z. Chen, G. Hefferman, T. Wei, H. He, Q. Yang, A hierarchical distributed fog computing architecture for big data analysis in smart cities, in: Proceedings of the ASE BigData & SocialInformatics 2015, ACM, New York, NY, USA, 2015, 28, 1–6, http://dx.doi.org/10.1145/2818869.2818898.
[41] S.M.A. Oteafy, H.S. Hassanein, IoT in the fog: A roadmap for data-centric IoT development, IEEE Commun. Mag. 56 (3) (2018) 157–163, http://dx.doi.org/10.1109/MCOM.2018.1700299.
[42] A. Aliyu, A.H. Abdullah, O. Kaiwartya, Y. Cao, J. Lloret, N. Aslam, U.M. Joda, Towards video streaming in IoT environments: Vehicular communication perspective, Comput. Commun. 118 (2018) 93–119, http://dx.doi.org/10.1016/j.comcom.2017.10.003.
[43] E. Badidi, H. Routaib, M. El Koutbi, Towards data-as-a-service provisioning with high-quality data, in: R. El-Azouzi, D.S. Menasche, E. Sabir, F. De Pellegrini, M. Benjillali (Eds.), Advances in Ubiquitous Networking, vol. 2, Springer, Singapore, 2017, pp. 611–623, http://dx.doi.org/10.1007/978-981-10-1627-1_48.
[44] A.V. Dastjerdi, H. Gupta, R.N. Calheiros, S.K. Ghosh, R. Buyya, Fog computing: Principles, architectures, and applications, in: R. Buyya, A.V. Dastjerdi (Eds.), Internet of Things, Morgan Kaufmann, 2016, pp. 61–75 (Chapter4), http://dx.doi.org/10.1016/B978-0-12-805395-9.00004-6.
[45] G. Optimization, Gurobi optimizer, 2018, https://www.gurobi.com/products/gurobi-optimizer. (Accessed 31 May 2018).
[46] IBM, Cplex optimizer, 2018, http://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer. (Accessed 31 May 2018).
[47] MathWorks, Matlab, 2018, http://www.mathworks.com/products/matlab.html. (Accessed 05 June 2018).

Firat Karatas received the B.S. and M.S. degrees from the Electrical and Electronics Engineering Department, Bilkent University, Ankara, Turkey, in 2008 and 2010, respectively. Now, he is currently pursuing the Ph.D. degree in the Computer Science Department of the same university.

Since 2012, he has been working in Meteksan Defence Ind. Inc., Ankara, Turkey, as a Digital Design Engineer. His current research interests include internet of things, resource optimization and data-driven applications.

Ibrahim Korpeoglu is a full professor in Department of Computer Engineering of Bilkent University, Ankara, Turkey. He received his Ph.D. and M.S. degrees from University of Maryland at College Park, both in Computer Science, in 2000 and 1996, respectively. He received his B.S. degree (summa cum laude) in Computer Engineering, from Bilkent University in 1994. Since 2002, he is a faculty member in Bilkent University. Before that, he worked in several research and development companies in USA including Ericsson, IBM T.J. Watson Research Center, Bell Laboratories, and Bell Communications Research (Bellcore). He received Bilkent University Distinguished Teaching Award in 2006 and IBM Faculty Award in 2009. His research interests include computer networks, wireless sensor networks and environment monitoring. He is a member of ACM and a senior member of IEEE.