HYBRID FOG-CLOUD BASED DATA DISTRIBUTION FOR INTERNET OF THINGS APPLICATIONS

A DISSERTATION SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

By
Fırat Karataş
September 2019

HYBRID FOG-CLOUD BASED DATA DISTRIBUTION FOR INTERNET OF THINGS APPLICATIONS

By Fırat Karataş

September 2019

We certify that we have read this dissertation and that in our opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

İbrahim Körpeoğlu(Advisor)

Ezhan Karaşan

Özgür Ulusoy

Ahmet Coşar

Ertan Onur

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

HYBRID FOG-CLOUD BASED DATA DISTRIBUTION FOR INTERNET OF THINGS APPLICATIONS

Fırat Karataş
Ph.D. in Computer Engineering
Advisor: İbrahim Körpeoğlu
September 2019

Technological advancements keep making machines, devices, and appliances faster, more capable, and more connected to each other. The network of all interconnected smart devices is called Internet of Things (IoT). It is envisioned that there will be billions of interconnected IoT devices producing and consuming petabytes of data that may be needed by multiple IoT applications. This brings challenges to store and process such a large amount of data in an efficient and effective way. Cloud computing and its extension to the network edge, fog computing, emerge as new technology alternatives to tackle some of these challenges in transporting, storing, and processing petabytes of IoT data in an efficient and effective manner.

In this thesis, we propose a geographically distributed hierarchical cloud and fog computing based IoT storage and processing architecture, and propose techniques for placing IoT data into its components, i.e., cloud and fog data centers. Data is considered in different types and each type of data may be needed by multiple applications. Considering this fact, we generate feasible and realistic network models for a large-scale distributed storage architecture, and propose algorithms for efficient and effective placement of data generated and consumed by large number of geographically distributed IoT nodes. Data used by multiple applications is stored only once in a location that is easily accessed by applications needing that type of data. We performed extensive simulation experiments to evaluate our proposal. The results show that our network architecture and placement techniques can be used to store IoT data efficiently while providing reduced latency for IoT applications without increasing network bandwidth consumed.

Keywords: Internet of Things, Fog Computing, Data Placement, Cloud Computing, Network Topology, Data Management.

ÖZET

NESNELERİN İNTERNETİ UYGULAMALARI İÇİN HİBRİT SİS-BULUT TABANLI VERİ DAĞITIMI

Fırat Karataş
Bilgisayar Mühendisliği, Doktora
Tez Danışmanı: İbrahim Körpeoğlu
Eylül 2019

Teknolojik gelişmeler makineleri, eşyaları ve cihazları daha hızlı, daha yetenekli ve birbirleriyle daha bağlı hale getirmeye devam etmektedir. Birbirlerine bağlı bu akıllı cihazların oluşturduğu ağ yapısı Nesnelerin İnterneti (IoT) olarak adlandırılmaktadır. Bu milyarlarca cihazın, birden fazla IoT uygulamasının ihtiyaç duyduğu ve duyabileceği petabaytlarca veriyi üretme ve/veya kullanma yeteneklerine sahip olacağı öngörülmektedir. Bu durum, büyük miktardaki bu verinin etkin ve verimli bir şekilde depolanması ve işlenmesi gibi zorlukları da beraberinde getirmektedir. Bulut bilişim ve onun son kullanıcılara yakınlaştırılmış versiyonu olan sis bilişim, IoT verilerinin verimli ve etkili bir şekilde nasıl taşınacağı, yerleştirileceği, saklanacağı ve işleneceği ile ilgili bu zorlukların bazılarının üstesinden gelebilmek için yeni yöntemler ortaya koymaktadır.

Bu tezde; coğrafi olarak dağıtık, hiyerarşik bulut ve sis bileşenlerini içeren bir IoT mimarisi ve oluşturulan büyük miktardaki IoT verisinin bu mimarinin bileşenlerine, bulut ve sis veri merkezlerine, yerleştirmek için yeni teknikler öneriyoruz. Verilerin sınıflandırılabileceği göz önünde bulundurulduğunda, birden fazla uygulama bu verileri kullanabilir. Bu bilgiye dayanarak, coğrafi olarak dağıtık IoT cihazlarının oluşturduğu ve kullandığı verileri, gerçeklenebilir ağ modellerinde etkin ve verimli şekilde veri merkezlerine yerleştirecek algoritmalar tasarlayıp, bunları tamsayı doğrusal modelleme yöntemi kullanılarak elde edilen optimum sonuçlarla karşılaştırıyoruz. Birden fazla uygulama tarafından kullanılan verileri, kopyalamadan ve o veriyi kullanan bütün uygulamalar tarafından rahatlıkla erişilebilecek bir merkezde saklıyoruz. Önerdiğimiz ağ mimarisi ve algoritmalar sayesinde; verilerin etkin ve verimli şekilde yerleştirilebileceğini, uygulamaların bu verilere bant genişliğini arttırmadan daha hızlı şekilde erişebilmesine olanak sağladığını yaptığımız kapsamlı simülasyon deneylerinin sonuçlarıyla doğruluyoruz.

Anahtar sözcükler: Nesnelerin İnterneti, Sis Bilişim, Veri Yerleştirme, Bulut

Bilişim, Ağ Mimarisi, Veri Yönetimi.

Acknowledgement

I would like to express my sincere gratitude to my advisor Prof. İbrahim Körpeoğlu for his guidance and patience. I am grateful for his encouragement and support during my hard times throughout the study, and I really appreciate that.

I would like to thank the members of the thesis monitoring committee, Prof. Ezhan Karaşan and Prof. Özgür Ulusoy for their precious comments and ideas during this long journey. I would also like to thank my thesis defense jury Prof. Ahmet Coşar and Assoc. Prof. Ertan Onur for accepting the invitation and taking a part in my defense jury.

I want to express my deepest gratitude to my family. Without their encouragement and patience I would not be here. I always sense their unlimited love and support in my heart. I also apologize from them for missing irreversible moments during this study.

I am very grateful to Prof. İbrahim Barışta from Medical Oncology Department of Hacettepe University during my hard times. I cannot accomplish this without him and my family.

I also want to thank my beloved friends who are always with me every time. I would especially thank Cem Mergenci for organizing the thesis monitoring committees together.

I am thankful to my supervisors and colleagues in Meteksan Defence Ind. Inc. for their support and encouragement.

The financial support of the Scientific and Technological Research Council of Turkey (TUBİTAK) through the BİDEB 2211 program is appreciated.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The idea of enabling communication between any pair of devices has led to the emergence of Internet of Things (IoT), which encompasses all techniques and technologies to interconnect all kinds of devices with each other via the Internet. This evolution has started first with the design of radio frequency identifiers (RFIDs) for small devices and things by using RFID technology and monitoring the states of tagged devices, and then continued with the developments in wireless sensor networks (WSN) [1].

There is an extensive amount of research and development going on in the IoT field [2]. As the types and the number of nodes constituting IoT are increasing every day, new and interesting IoT applications are envisioned and built [3] as well. Not only the applications, but also the different types of technologies depending on IoT are increasing. There are a lot of studies explaining the evolution in IoT applications and technologies with future trends and research challenges [4–8].

Advances in IoT and development of new IoT products enable new kinds of IoT concepts. For example, the advances make wearable smart devices connected to the Internet popular, and now they are named as Wearable IoT (WIoT) [9]. Also, it is expected in the near future that the concept of smart cities [10] will be implemented and deployed, and there are a lot of research activities going on to

tackle the related challenges [11–13].

While IoT technologies were evolving, new communication and networking technologies supporting IoT have been designed as well. One of these is the low-power wide area networks (LPWAN) technology [14, 15], which is a long-range low-rate wireless communication technology that consumes low power. With upcoming communication technologies, releasing the potential of IoT will become faster. For example, 5G technology will increase the communication speed and coverage between devices to support high-rate and high-quality demanding applications [16].

Various applications can use IoT technology nowadays. Some of these applications include, but not limited to, crowd-sensing [17], smart home [18] and smart grid [19]. Since the number of potential applications is growing steadily, a classification can help to see the trends and analyze the requirements, and Scully [20] does this by classifying all IoT applications into ten different groups. These applications are reviewed and explained in various papers, such as [21–23], and one of the common problems of these applications is handling big data.

Since the number nodes forming IoT can increase exponentially, the volume of data generated and consumed by these nodes can increase to an unprecedented level [24]. Storing and processing such a big volume of data is a daunting task [25]. This becomes especially challenging when IoT nodes are geographically distributed to a very large region [26].

Distributed cloud computing technology is an attractive alternative to store and process large volumes of data [27]. Cloud computing can be used to assign complicated and resource hungry tasks to capable data centers distributed around the world [28]. New architectures are needed, however, to integrate cloud computing with IoT [29]. 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 [30]. 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 [31–34].

Integration of cloud and fog computing paradigms together with IoT requires the adaptation of the applications to this approach. One problem to consider is how to place and use application data in the components of the distributed and centralized cloud infrastructure [35,36]. Such an infrastructure will enable a lot of new data-intensive applications, and these applications may share the generated IoT data. The dependency relationship of the applications to the data becomes critical in designing data placement strategies.

In this thesis, we focus on this problem of designing a cloud and fog based IoT data storage architecture and related data placement methods on this infrastructure. We first propose a hybrid 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 hybrid architecture are IoT devices and they are the least capable components available in the architecture. Primary services offered by the cloud and fog data centers in the proposed architecture is data-as-a-service (DaaS) [29], 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 [37]. 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 demand for data and this may become an important optimization factor, since the amount of data shared by applications may be quite large. If each application stores the needed data separately to maximize its benefits, which we name as *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 huge amount of data efficiently without disrupting application requirements.

To overcome the inefficient data storage 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 which is optimum for multiple applications requiring it. We call this approach as *data-centric approach*. Based on this approach, we propose methods about where to store different types of data efficiently so that the applications can reach their required data with the lowest feasible latency.

Part of this thesis is based on our work in [38], which is proposing consideration of IoT data in different types, and at the same time proposing methods to place different types of data efficiently and effectively 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 the performance, which is important from the DaaS providers point of view.

Since there is no known deployed or well-known verification environment to test fog based IoT algorithms, we also developed a model for hybrid fog-cloud based IoT networks. By using this model we tested our heuristic algorithms.

We can summarize the contributions of our thesis as follows:

1. We propose a hybrid hierarchical fog-cloud computing data center network architecture involving IoT nodes for storing and processing IoT data efficiently.

2. We propose the consideration of IoT data in various and well-defined types, based on the consideration that multiple IoT application may need to access the same data. This allows efficient data storage, since data can be stored only in one place and can be shared. While doing this, we also care about not increasing network overhead and satisfying delay requirements.

3. According to the proposed data classification architectural model, we develop a mathematical model for average data access latency that applications encounter, and provide an integer programming solution.

4. To achieve the best performance according to the given analytical model without using linear solvers, we propose efficient data placement heuristic algorithms to place the data in the mesh network of fog computing units, which can provide efficient storage, low latency and reduced network overhead compared to application-centric approach.

5. Since there is no known deployed fog-cloud based IoT architecture, we propose a method to model hybrid fog-cloud computing networks, and use this model for the evaluation of our data placement algorithms.

1.1 Thesis Outline

The rest of this thesis is organized as follows. In Chapter 2, we give the state-of-the-art and related work in IoT and handling of IoT data. In Chapter 3, we specify the IoT data placement problem and propose a hybrid and hierarchical fog-cloud based IoT storage architecture. In Chapter 4, we explain our analytical model and give linear programming solution for the data placement problem in the proposed storage architecture. In Chapter 5, we propose heuristic data placement algorithms that can be used in the proposed architecture when the network of fog and cloud data centers gets larger. In Chapter 6, we explain the

methods and our model for the simulated IoT network, which is required for the evaluation of the algorithms. In Chapter 7, we present the results of our extensive simulation experiments done based on the proposed IoT network model. Finally, we give our conclusions and future work in Chapter 8.

Chapter 2

Related Work

Fog computing, an extension of cloud computing, tries to push clouds through edges of the network, and can be employed in IoT networks to store and process IoT data more effectively and efficiently. All of the work done in the literature related to cloud and fog computing together with IoT have significant importance to this research. On top of these studies, we focus on an important challenge of big data placement in networks consisting of these elements where some of them have limited resources (especially in IoT nodes). Under the light of these, we can group fog-cloud enabled IoT related research studies into three major groups:

1. Network architecture and use cases.
2. Allocation of available resources.
3. Data placement strategies.

2.1 Network Architecture

The first group of related work consists of architectural and theoretical work done in cloud and fog enabled IoT. In one of these works, Bonomi et al. [26] explain the

interplay of data in combination with IoT and fog-cloud computing paradigms, and give some architectural structures for applications running in IoT nodes.

Since deploying and designing a smart city from network perspective is challenging and competitive; verifying the algorithms and use cases of the smart city requires effort. For making these easier Santos et al. define a framework and testbed called City of Things (CoT) [39] and deploy it in Antwerp, which enables researchers to test possible IoT applications in smart cities. In another work, Zanella et al. [21] 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. [40] explain a hypothetical network structure for smart city applications using the 5G network. Most of the network architectures enabling fog computing concept require intelligent gateways and routers in the edges. As an example of these, including, but not limited to, Aazam and Huh explain the smart gateway concept in [41], and Jutila gives an efficient edge router in [42].

2.2 Resource Allocation

The second group of related work is resource provisioning or allocation, which is inevitable where resources are limited and lots of demanders such as 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 examples of this research are including, but not limited to [43–46].

An important resource in IoT nodes that needs to be used carefully is the network bandwidth, which is usually shared by a lot of applications. Angelakis et al. [47] propose an allocation model for heterogeneous resource demands by

considering activation and utilization metrics of available network interfaces in IoT devices. Tsai [48] proposes a network resource allocation algorithm for IoT devices, called SEIRA, based on search economics for exploring solution space for getting close to the optimum. Lera et al. [49] consider reducing network usage by placing IoT data on fog computing nodes according to centrality indices. However, they do not consider the characteristics of the distributed applications running in the network, such as their quality-of-service requirements, data access patterns, or where they are running.

In another group of the resource allocation studies, researchers want to answer how IoT modules should be decoupled into fog and cloud computing nodes such that the quality-of-service requirements are met. In one of these works, Taneja and Davy consider partitioned application module layers and aim to place each of these modules to computing nodes efficiently [50]. In another work, Rezazadeh et al. [51] try to place IoT modules in fog and cloud nodes using simulated annealing. The same group propose a heuristic called LAMP to decrease the latency applications encounter by placing different IoT application modules to fog and cloud computing nodes [52]. Natesha and Guddeti use another heuristic called First-Fit Decreasing (FFD) aiming again to decrease the latency applications encounter together with the power consumption of computing nodes [53].

Resource provisioning is also another very important problem in more limited fog computing data centers. Tocze and Nadjm-Tehrani [54] 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 the 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. [55] 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. [56] 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. [57] 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. [58] describe MIST, which is a less dense communication scheme for fog computing, and model the 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 the reduction of their nonlinear MIST model to a linear one. Skarlat et al. [59] 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. [60] 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. [61] 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 IoT devices. 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 increase, data placement for these devices 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 [62] 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. [63] discuss the advantages of the 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 [64] 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 [65]. 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 [66].

Chapter 3

Proposed Hybrid Fog-Cloud Based IoT Data Placement System Architecture

We focus on networks which are composed of cloud and fog computing data centers together with the geographically distributed and less capable IoT nodes. By grouping and using these incapable nodes in each group with harmony, complex applications may run and each of these may need some sort of data produced by other nodes or groups. As the number of these less capable nodes increase, processing, distributing and storing data become problematic. Therefore, we try to answer the question of how to distribute and store data required by the geographically distributed nodes in this section.

In this chapter, we start with defining our hierarchical hybrid fog-cloud based network architecture (Chapter 3.1) which consists of both fog and cloud data centers together with less capable IoT nodes for overcoming distribution and storage problems. On top of the considered network architecture, we explain a placement strategy which we call *data-centric approach* (Chapter 3.2). And finally, we present agents which are called *Data Profiler* and *Data Classifier* needed for this solution to work in the defined network efficiently (Chapter 3.3).

3.1 Hybrid Fog-Cloud Based Network Architecture

The network architecture starts with considering the hierarchical structure of geo-distributed IoT data that is mentioned in the work of Bonomi et al. [26]. In their geo-distributed structure, latency increases as data goes from IoT nodes to cloud computing centers and this creates 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 these in mind, 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 [63]. 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 with star topology (see Figure 3.1). There may be more than one FC in an FCU. All FCUs and CCs are connected 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.

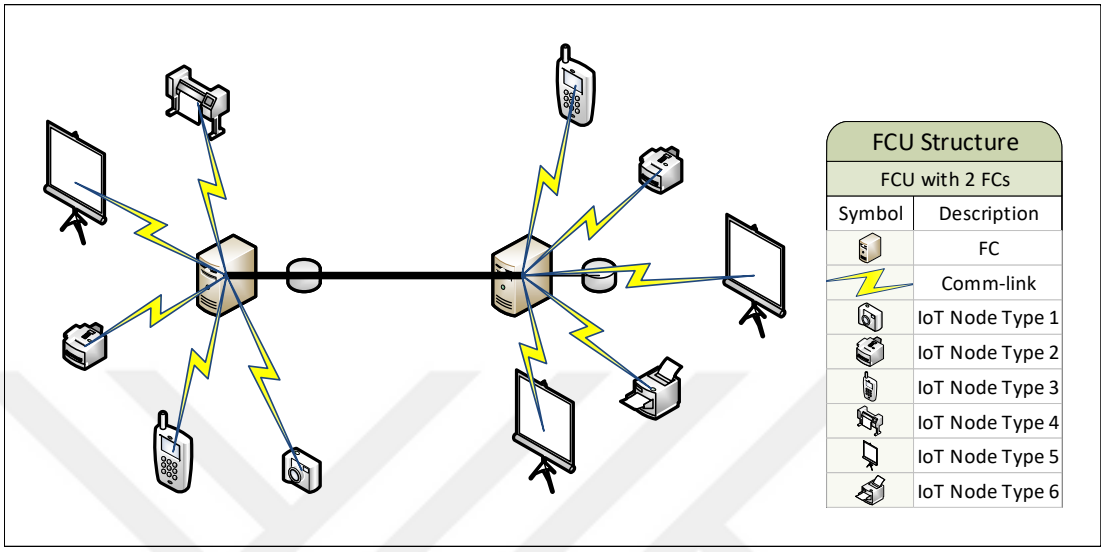


Figure 3.1: An example FCU consisting of 2 FCs and 10 IoT nodes.

3.2 Data-Centric IoT Data Placement Strategy

In Section 3.1, we define the network architecture which only shows the possible places of data, in this section we want to explain the data placement problem and present a solution strategy. 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 the corresponding 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 IoT applications in need. 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 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 the 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 with 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 another application running. It also requires traffic data as in the case of health services and if the application-centric data placement is used, the traffic data will be duplicated. By using the data-centric approach and classifying data into types and placing them accordingly, we 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 [67] 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 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 together can measure and derive the data characteristics of IoT nodes, and notify the central mechanism for data placement procedures (see Figure 3.2). 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 the outcome of the agents, 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 the data needs. Designers can use a publish-subscribe mechanism for accessing the 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 the reduction of storage costs. Regarding that point of view, the data-centric approach can perform better than the application-centric data placement approach dramatically.

Decreasing the storage costs by using the data-centric approach is an important benefit. 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 need 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

[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 0.4msec for a 60km distance. If small amount of data is assumed to be 1KB, then its processing time becomes in the order of nanoseconds when a simple RAID 0 architecture used with two SSDs whose read speeds are 500MB/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.

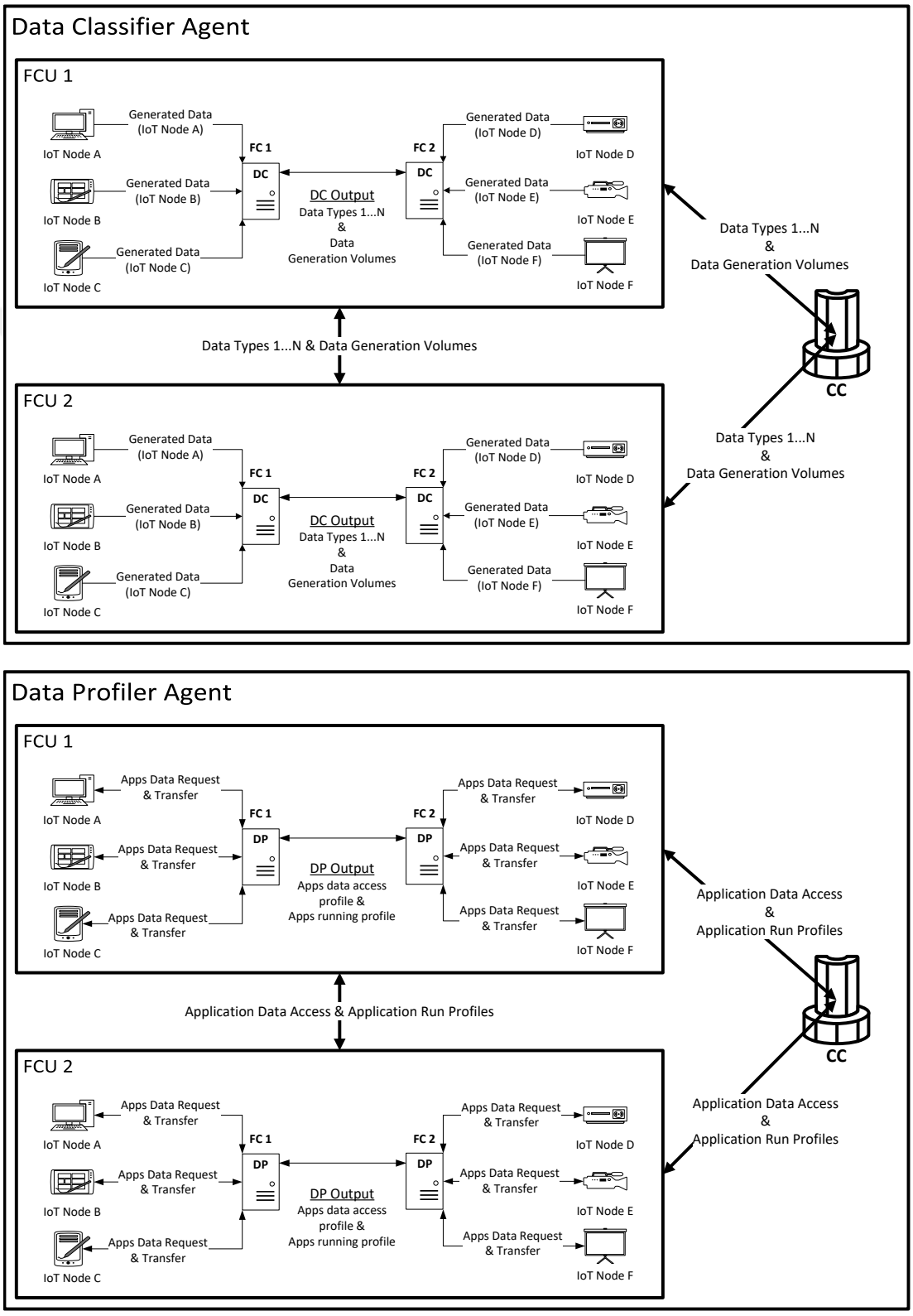


Figure 3.2: Example scenarios of Data Classifier & Data Profiler agents in the proposed architecture.

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 the geographically distributed FCUs needs to be minimized without duplicating the data. We formulate this optimization problem and give its linear model in Chapter 4.

Under the scope of our model, the mobility of 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.

3.4 Summary

In this chapter, we start with explaining a network model consisting of cloud computing data centers, IoT nodes and fog computing data centers in between. We merge the geographically close fog computing data centers and IoT nodes, and name each of these merged groups as fog computing unit (FCU).

By using the available storage resources of FCUs with harmony, we want to decrease storage burden on cloud computing data centers. For doing it so, we need to split data into well-known types and store in types. We call it as the data-centric placement approach.

Finally, we describe two agents required to make data-centric placement approach feasible. One of the agents is responsible for splitting data and we name it data classifier (DC), and the second agent is responsible for monitoring the data access and running characteristics of the applications in FCUs.

Chapter 4

IoT Data Placement Problem in Hybrid Fog-Cloud Based Architecture

In the previous chapter, we explain a structural solution for data placement problem, and in this chapter we present the analytical model of the problem. As mentioned at the end of Section 3.3, when multiple applications and data types exist in network, data placement turns into a problematic task. By using the outputs of agents defined, we can reduce it to an optimization problem and give the solution in Section 4.1. According to the optimization model we define any of the commercial linear solvers such as Gurobi [68] or CPLEX [69] can give the optimum solution of the problem.

One of the important obstacles for developing new IoT applications is the data to be used. Since we consider data in well-defined types, new applications can be developed easily using existing data types. Under the light of this circumstance, we foresee the gap between number of applications and available data types increases. So we need a metric to define the relation between applications and data types. We name this metric excess use and explain it in Section 4.2.

We conclude this chapter by explaining the linearity of the proposed model with a numerical example in Section 4.3.

4.1 Analytical Model of Data Placement in Hybrid Fog-Cloud Architecture

As mentioned in Section 3.1, 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 of these IoT and FC nodes in that region are called an FCU, and data is generated/consumed or both in these. Generated data is placed either in FCUs or CCs, and applications running in FCUs use the generated 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. In other words applications can share the same data types.

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

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

Our system model has the following parameters:

$\boldsymbol{M}$	$\rightarrow$	Total number of CCs
$\boldsymbol{N}$	$\rightarrow$	Total number of FCUs
$\boldsymbol{D}$	$\rightarrow$	Total number of different data types existing
$\boldsymbol{A}$	$\rightarrow$	Total number of different applications running
$\boldsymbol{L}$	$\rightarrow$	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 the geographical locations of data centers. Placement of the elements in $\boldsymbol{L}$ 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 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 the IoT nodes (i.e., in FCUs), and which data types they access and how often. We define these frequencies as follows:

$\boldsymbol{AR}$	$\rightarrow$	Application running frequencies in the FCUs
$\boldsymbol{AF}$	$\rightarrow$	Data access frequencies of the 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 running 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 the 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:

$$\begin{aligned} \boldsymbol{GV} &\rightarrow \text{Data generation volume for each type of data in each FCU} \\ \boldsymbol{DG} &\rightarrow \text{Total data generation volume for each type of data} \\ \boldsymbol{P} &\rightarrow \text{Final placement matrix, where each data type is stored} \end{aligned}$$

where $\boldsymbol{GV}$ is an $(\boldsymbol{N} \times \boldsymbol{D})$ 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 is either 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 Equation 4.1.

$$dg_i = \sum_{j=1}^{N} gv_{j,i} \tag{4.1}$$

$\boldsymbol{P}$ is a binary matrix because partial data placement and replication are not allowed, and it has to satisfy Equation 4.2.

$$\sum_{k=1}^{M+N} p_{i,k} = 1, \forall i \in D \tag{4.2}$$

According to the architecture we describe in Section 3.1, FCs have limited storage capacities, since they are small-scale versions of CCs. Therefore, we define a variable for denoting the storage capacity constraint for each FCU, and there is also another variable which indicates how much of the available capacity in an FCU is used:

$$\begin{aligned} \boldsymbol{SC} &\rightarrow \text{Storage capacity of FCUs} \\ \boldsymbol{UC} &\rightarrow \text{Used capacity of FCUs} \end{aligned}$$

where $\boldsymbol{SC}$ and $\boldsymbol{UC}$ are $(1 \times \boldsymbol{N})$ vectors. After the placement of all the available data types, we need to satisfy the following equation:

$$uc_i \leq sc_i\,, \quad i \in \{1, 2, \ldots, N\} \tag{4.3}$$

Referring back to the verbal definition of the problem in Chapter 3, the goal is to minimize the average latency that applications encounter while obtaining

Table 4.1: 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

the required data from the data centers (fog or cloud centers) where it is stored. Table 4.1 summarizes all of the notation used for describing the model, and Equation 4.4 describes the average latency applications encounter while accessing data by using these variables.

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

$$= \sum_{i=1}^{A} \frac{\sum_{k=1}^{N} \sum_{y=1}^{D} ar_{k,i} \mathbf{sgn}(af_{i,y})(\sum_{z=1}^{M+N} l_{z,(M+k)} p_{y,z})}{\sum_{j=1}^{D} \mathbf{sgn}(af_{i,j}) \sum_{w=1}^{A} \sum_{q=1}^{N} ar_{q,w}} \quad (4.4)$$

The denominator of Equation 4.4 is a scaling factor depending on the application running frequency in the FCUs, and the numerator is total latency

applications encounter while gathering the required data. To explain the details, "$\mathbf{L} \times \mathbf{P}$" denotes the latency application encounter while reaching the required data types where they reside, and it is multiplied by the sign function of the data access frequencies of the application which is "$\mathbf{sgn}(\mathbf{AF})$". The output of this multiplication gives the latency for an application to access all the required data types, and finally it is multiplied by the application's running frequency "$\mathbf{AR}$" in an FCU which is obtained from the output of DP agent.

$\boldsymbol{sgn}(.)$ is the sign function with the following definition:

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

Why we use the sign function in the formulation is caching. As we mention in Chapter 3, for a considered time-interval applications need a small amount of data and when they gather required data type, they cache it inside the FCU.

We can reduce the Equation 4.4 into a matrix by form using the matrices $\boldsymbol{AR}$, $\boldsymbol{AF}$, $\boldsymbol{L}$ and $\boldsymbol{P}$. Then we have the following equations in matrix form:

$$\boldsymbol{DL} = \boldsymbol{P} \times \boldsymbol{L} \tag{4.5}$$

$$= \begin{bmatrix} \overrightarrow{dl_1} & \overrightarrow{dl_2} & \dots & \overrightarrow{dl_{M+N}} \end{bmatrix}$$

$$\boldsymbol{FL} = \begin{bmatrix} \overrightarrow{dl_{M+1}} & \overrightarrow{dl_{M+2}} & \dots & \overrightarrow{dl_{M+N}} \end{bmatrix} \tag{4.6}$$

$$\boldsymbol{DD} = \mathbf{sgn}(\boldsymbol{AF}) \tag{4.7}$$

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

$$\boldsymbol{AL} = \boldsymbol{DD} \times \boldsymbol{FL} \tag{4.8}$$

$$\boldsymbol{ARL} = \boldsymbol{AL} \circ \boldsymbol{AR}^{T} \tag{4.9}$$

$$\boldsymbol{SF} = \begin{bmatrix} \overrightarrow{sf_1} & \overrightarrow{sf_2} & \dots & \overrightarrow{sf_N} \end{bmatrix} \tag{4.10}$$

$$\overrightarrow{\boldsymbol{sf_i}} = \|\boldsymbol{AR}\|_{1,1} \times \begin{bmatrix} \left\|\overrightarrow{dd_1}\right\|_1 \\ \left\|\overrightarrow{dd_2}\right\|_1 \\ \vdots \\ \left\|\overrightarrow{dd_2}\right\|_A \end{bmatrix} \tag{4.11}$$

$$\boldsymbol{AAL} = \boldsymbol{SF} \circ \boldsymbol{ARL} \tag{4.12}$$

$$\text{Average Latency} = \|\boldsymbol{AAL}\|_{1,1} \tag{4.13}$$

In Equation 4.5, $\boldsymbol{DL}$ 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 $\boldsymbol{DL}$ 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 $\boldsymbol{N}$ column vectors to form another matrix $\boldsymbol{FL}$ whose size is $(\boldsymbol{D} \times \boldsymbol{N})$, we obtain the latency of each FCU for reaching each data type (Equation 4.6). From the applications point of view, data dependencies are important and it is shown by $\boldsymbol{DD}$ 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 $\boldsymbol{i}$ on the data types. When the matrices obtained in Equation 4.6 and Equation 4.7 are multiplied (Equation 4.8), we obtain the latency matrix $\boldsymbol{AL}$ 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 the FCUs are not taken into account. In order to consider the effect of these profiled running frequencies on latencies, we obtain the $\boldsymbol{ARL}$ matrix by using the Hadamard product operator $(\circ)$ in Equation 4.9. This operation is an element-wise product of the entries in matrices $\boldsymbol{AL}$ and $\boldsymbol{AR}^T$, and $\boldsymbol{ARL}$ is the weighted sum of the latencies applications encounter in which FCU they run. For normalizing the weighted latencies obtained by $\boldsymbol{ARL}$ matrix, we define a scaling factor $\boldsymbol{SF}$ in Equations 4.10 and 4.11. In Equation 4.11, $\|.\|_1$ and $\|.\|_{1,1}$ denote L_1 norms of a vector and a

matrix respectively. $L_{p,q}$ norm of an $(m \times n)$ matrix $\boldsymbol{A}$ is defined as:

$$\|A\|_{p,q} = \left[\sum_{j=1}^{n} \left(\sum_{i=1}^{m} |a_{ij}|^p \right)^{(p/q)} \right]^{(1/p)} \tag{4.14}$$

If the scaling factor and the weighted sum latency matrices are multiplied element-wise, we obtain the $\boldsymbol{AAL}$ matrix, whose size is $(\boldsymbol{A} \times \boldsymbol{N})$ (Equation 4.12). It denotes the average weighted latency of each application running on each FCU. As described in Chapter 3, the aim is to minimize average latency that applications encounter. Therefore, if the sum of each element in matrix $\boldsymbol{AAL}$ denoted as $\|.\|_{1,1}$ is minimized, then the goal is achieved. Equation 4.13 is the matrix form of the Equation 4.4.

Equations 4.15 to 4.18 give the integer programming model of the problem.

Minimize:

$$\sum_{i \in A} \frac{\sum\limits_{k \in N} \sum\limits_{y \in D} ar_{k,i} \mathbf{sgn}(af_{i,y}) \left(\sum\limits_{z \in M \cup N} l_{z,(M+k)} p_{y,z} \right)}{\sum\limits_{j \in D} \mathbf{sgn}(af_{i,j}) \sum\limits_{w \in A} \sum\limits_{q \in N} ar_{q,w}} \tag{4.15}$$

Subject to:

$$p_{i,j} \in \{0, 1\} \tag{4.16}$$

$$\sum_{j \in M \cup N} p_{i,j} = 1, \forall i \in D \tag{4.17}$$

$$\sum_{i \in D} dg_i \times p_{i,(M+j)} \leq sc_j, \forall j \in N \tag{4.18}$$

In integer programming model, Equation 4.15 is the objective function, which is same as the cost function defined in the Equation 4.4. Equation 4.16 ensures that partial data placement and replication are not allowed, while Equation 4.17 guarantees every data type is placed. Finally, Equation 4.18 indicates that the storage capacities of the FCUs are not exceeded.

4.2 Relation Between Applications and Data Usage

We envision that multiple applications may need the same type of data. Since our approach to efficiently store, access and process IoT data involves considering data in different types,e.g. traffic data, health data, applications can use any of these available data types to run correctly and this requires the sharing of data. In the emergency application given 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 the navigation service, only traffic data is required, but for the health service, 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. The application-centric data placement approach stores the data for each application separately, but the data-centric approach that our architecture is using enables the sharing of the stored data by multiple applications. The question is how efficient the proposed architecture is instead of using the 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 required data types by the applications. But this is not a compact form to express the sharing amount of data among the applications (i.e., data overlap ratio), so we need to define a metric to indicate how many applications require a specific data type.

We use a simple bipartite graph to show the relationship between the applications and the data types. The vertices are partitioned into two sets: applications and data types. The edges of the bipartite graph denote the dependencies of applications on the data types, which are the non-zero elements of **AF** matrix. An example of this bipartite graph is given in Figure 4.1.

If we consider the example graph, 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

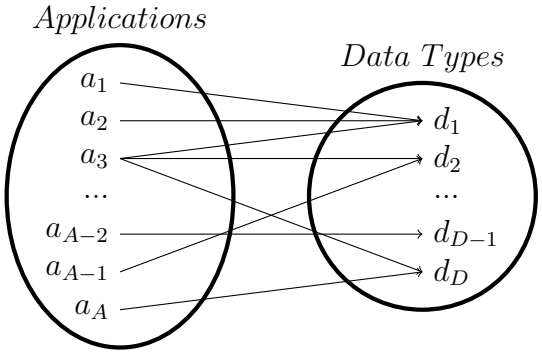


Figure 4.1: An example of application - data type relation graph.

in such a case that some of the data types are required by multiple applications, and if the application-centric data storage is used, the data would be replicated and the storage cost would increase.

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. Therefore, each stored data type is needed by at least one application. This gives the baseline definition of data excess use (sharing amount or overlap ratio) and is defined in Equation 4.19.

$$\textbf{ExcessUse} = \frac{\sum_{i=1}^{A} \sum_{j=1}^{D} \textbf{sgn}\left(af_{ij}\right)}{D} \tag{4.19}$$

Equation 4.19 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.

$$\textbf{ExcessUse} = 1 \tag{4.20}$$

If Equation 4.20 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.

$$\mathbf{ExcessUse} > 1 \tag{4.21}$$

If Equation 4.21 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 use depends on **AF** matrix (Equation 4.19) whose size is defined by **A** and **D**. Therefore, the values of excess use is related with both **A** and **D** parameters, and there exists three possible conditions:

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

   Lemma 4.2.1. *If this is the case then:*

   $$\mathbf{ExcessUse} > 1$$

   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 the data types in the network. If data and applications are assumed as two different 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. Since the number of non-zero elements in **AF** is **A**, the following equation has to be satisfied:

   $$\mathbf{ExcessUse} = \mathbf{A}/\mathbf{D}$$
   $$\mathbf{A} > \mathbf{D} \rightarrow \mathbf{ExcessUse} > 1.$$

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

   $$\mathbf{ExcessUse} = \mathbf{A}.$$

   □

2. $\mathbf{A} = \mathbf{D}$: This case is a probable case in a scenario, since there is no limitation on application and data counts.

   Lemma 4.2.2. *If this is the case then:*

   $$\mathbf{ExcessUse} \ \geq \ 1$$

   then the 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

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

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

   $$\mathbf{ExcessUse} \ = \ \mathbf{A}.$$

   □

3. $\mathbf{A} < \mathbf{D}$: 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 the available data types.

   Lemma 4.2.3. *If this is the case then:*

   $$\mathbf{ExcessUse} \ \geq \ 1$$

   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 $\mathbf{D}$ non-zero elements in $\mathbf{AF}$ matrix. So excessive usage satisfies

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

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

   $$\mathbf{ExcessUse} \ = \ \mathbf{A}.$$

   □

Given cases are the theoretical limits of excess use in the proposed architecture and help us during the setup of our simulations.

4.3 Proof of the Linearity of Mathematical Model

Formal problem formulation given in Section 4.1 with Equation 4.4 seems complicated, and it can easily be confused with quadratic optimization although it is linear. In this section we want to review the formulation in detail and prove it is a linear optimization problem with an example.

Let's start with Equation 4.4. In the denominator of the formal problem definition, there exists a constant when we collect the outputs of DP agent from all FCUs. It is a normalization factor for running frequencies of all applications in all FCUs, and it is given in Equation 4.22. It is the sum of all elements in the **AR** matrix and for the sake of simplicity let us denote it with **C**.

$$C = \sum_{w=1}^{A} \sum_{q=1}^{N} ar_{q,w} \tag{4.22}$$

After changing the summation with constant **C**, the formal form of the problem becomes the following:

$$\frac{1}{C} \sum_{i=1}^{A} \frac{\sum_{k=1}^{N} \sum_{y=1}^{D} ar_{k,i} \mathbf{sgn}(af_{i,y}) (\sum_{z=1}^{M+N} l_{z,(M+k)} p_{y,z})}{\sum_{j=1}^{D} \mathbf{sgn}(af_{i,j})} \tag{4.23}$$

The denominator of Equation 4.23 is a column sum of a binary matrix $\mathbf{sgn}(AF)$ and each element of this sum indicates total number of different data types requird by the corresponding application. We can easily replace this summation with a row vector $\overrightarrow{\mathbf{B}}$ whose elements are the column sums:

$$\overrightarrow{\mathbf{B}} = \sum_{j=1}^{D} \mathbf{sgn}(af_{i,j}) \tag{4.24}$$

And finally let us define a new matrix called **SAF**:

$$\mathbf{SAF} = \mathbf{sgn}(\mathbf{AF}) \tag{4.25}$$

The formal form of the equation becomes the following:

$$\frac{1}{C}\sum_{i=1}^{A}\frac{1}{b_i}\sum_{k=1}^{N}\sum_{y=1}^{D} ar_{k,i}saf_{i,y}(\sum_{z=1}^{M+N} l_{z,(M+k)}p_{y,z}) \tag{4.26}$$

It is definite based on Equation 4.26 that only variable left after obtaining the outputs of DP and DC agents in the network is placement matrix **P**. When the outputs of agents are available, **AF**, **AR** matrices, constant **C** and the row vector **B** are calculated. Also, latency matrix **L** has been already known in the beginning, and this concludes that there exists only one variable left to be decided which is the placement matrix **P**. Since the defined cost function is not in the quadratic form according to decision variable **P**, our formulation is linear.

Although we show the decision variable (placement matrix **P**) is not multiplied with itself, we need to check that there is no multiplication between the elements of the placement matrix. When we expand the equation element-wise, we see that there is no multiplication between the elements of the decision matrix, and this makes it possible to obtain optimal solution with commercial state of the art linear solvers.

Let us consider the following example to expand the equation in element-wise to see the linearity of the problem. There exists 5 applications using 3 data types running in 2 FCUs, and there is an CC between these 2 FCUs. This example is given graphically in Figure 4.2.

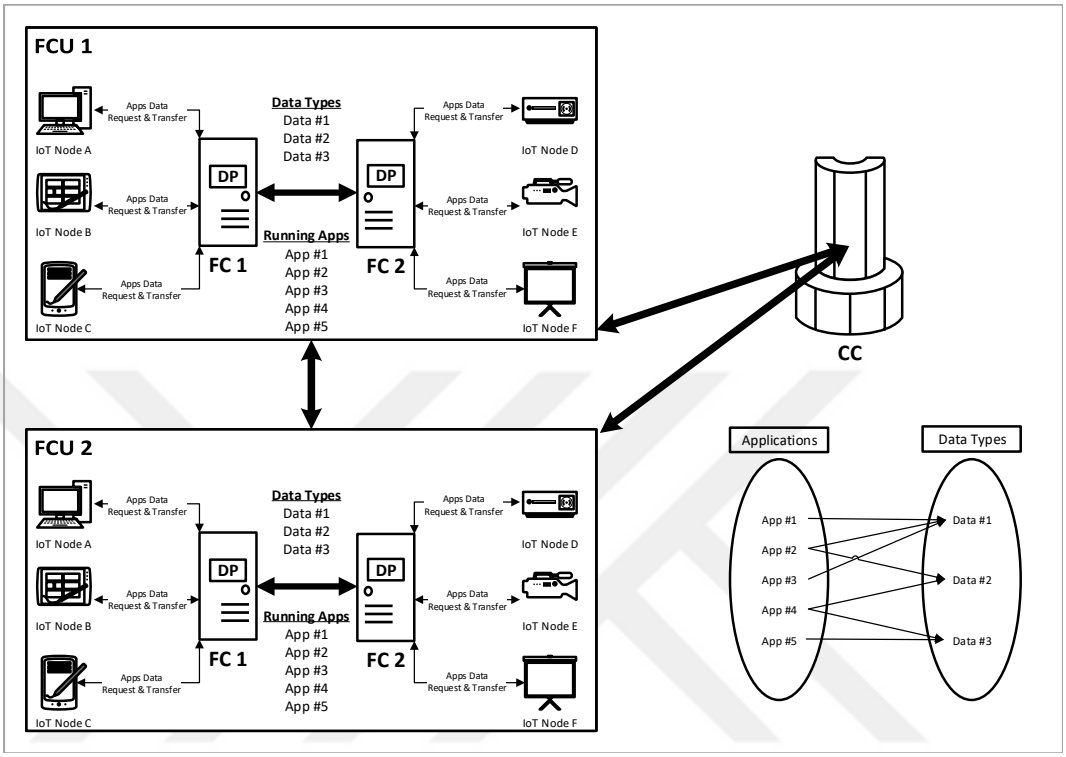


Figure 4.2: An example of network & data dependency graph (2 FCUs, 1 CC, 5 applications and 3 data types).

If we put the numbers in Equation 4.26, we get the following:

$$\frac{1}{C}\sum_{i=1}^{5}\frac{1}{b_i}\sum_{k=1}^{2}\sum_{y=1}^{3} ar_{k,i} saf_{i,y}(\sum_{z=1}^{3} l_{z,(1+k)} p_{y,z}) \tag{4.27}$$

After writing all the elements together with constant **C** and column sum vector **B** in the equation, we obtain the following:

$$\frac{1}{C}\sum_{i=1}^{5}\frac{1}{b_i}\sum_{k=1}^{2}\sum_{y=1}^{3} ar_{k,i}saf_{i,y}(\sum_{z=1}^{3} l_{z,(1+k)}p_{y,z})$$

$$= \frac{1}{C}\sum_{i=1}^{5}\frac{1}{b_i}\sum_{k=1}^{2}\sum_{y=1}^{3} ar_{k,i}saf_{i,y}(l_{1,(1+k)}p_{y,1} + l_{2,(1+k)}p_{y,2} + l_{3,(1+k)}p_{y,3})$$

$$= \frac{1}{C}\sum_{i=1}^{5}\frac{1}{b_i}\sum_{k=1}^{2} ar_{k,i}(saf_{i,1}l_{1,(1+k)}p_{1,1} + saf_{i,1}l_{2,(1+k)}p_{1,2} + saf_{i,1}l_{3,(1+k)}p_{1,3}+$$
$$saf_{i,2}l_{1,(1+k)}p_{2,1} + saf_{i,2}l_{2,(1+k)}p_{2,2} + saf_{i,2}l_{3,(1+k)}p_{2,3}+$$
$$saf_{i,3}l_{1,(1+k)}p_{3,1} + saf_{i,3}l_{2,(1+k)}p_{3,2} + saf_{i,3}l_{3,(1+k)}p_{3,3})$$

$$= \frac{1}{C}\sum_{i=1}^{5}\frac{1}{b_i}(ar_{1,i}saf_{i,1}l_{1,2}p_{1,1} + ar_{1,i}saf_{i,1}l_{2,2}p_{1,2} + ar_{1,i}saf_{i,1}l_{3,2}p_{1,3}+$$
$$ar_{1,i}saf_{i,2}l_{1,2}p_{2,1} + ar_{1,i}saf_{i,2}l_{2,2}p_{2,2} + ar_{1,i}saf_{i,2}l_{3,2}p_{2,3}+$$
$$ar_{1,i}saf_{i,3}l_{1,2}p_{3,1} + ar_{1,i}saf_{i,3}l_{2,2}p_{3,2} + ar_{1,i}saf_{i,3}l_{3,2}p_{3,3}$$
$$ar_{2,i}saf_{i,1}l_{1,3}p_{1,1} + ar_{2,i}saf_{i,1}l_{2,3}p_{1,2} + ar_{2,i}saf_{i,1}l_{3,3}p_{1,3}+$$
$$ar_{2,i}saf_{i,2}l_{1,3}p_{2,1} + ar_{2,i}saf_{i,2}l_{2,3}p_{2,2} + ar_{2,i}saf_{i,2}l_{3,3}p_{2,3}+$$
$$ar_{2,i}saf_{i,3}l_{1,3}p_{3,1} + ar_{2,i}saf_{i,3}l_{2,3}p_{3,2} + ar_{2,i}saf_{i,3}l_{3,3}p_{3,3})$$

...

Equation continues, and it is obvious that there exists no multiplication between any of the decision variables which concludes that the proposed formulation is linear.

4.4 Summary

We start this chapter with a mathematical model of the data placement problem explained in Chapter 3. We can obtain optimal solutions by using this model with any of the linear-solvers available. As it can be clear in the performance evaluation part that finding optimal solution become time-consuming when the values of the parameters increase, so we propose four heuristic algorithms as an alternative to this optimization model in Chapter 5.

Since we envision that the gap between the number of different applications and data types increases, we define a metric called excess use to be used for defining the dependencies of applications and data types.

At the end of this chapter, we expand the complex mathematical equation with a numerical example to prove its linearity. This guarantees when the mathematical model is solved with linear solvers, the output is optimal, and the best decisions given in the performance evaluation part are comprised of these results. In other words, we evaluate our proposed algorithms according to the linear model given in this chapter.

Chapter 5

Algorithms for Data Placement Problem in Hybrid Fog-Cloud Based Architecture

Chapter 4 reduces the problem in our proposed architecture to a binary integer programming (BIP) problem, thus we can use any state of the art commercial solvers such as Gurobi [68] or CPLEX [69] to solve the problem. The solution time for these solvers increase exponentially as the number of data centers, data types and running applications increase.

To overcome this, we propose four heuristic algorithms for reducing the defined cost function and obtaining a solution close to the optimal one obtained by solvers. In all algorithms, we assume that data generation volumes, data access patterns and running frequencies of the applications are known a priori. As explained in Section 3.3, these values are tracked by DC and DP agents running 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, monitor all data generation and usage characteristics, and provide the necessary information for constructing **GV**, **AR** and **AF** matrices.

We propose four algorithms and group in two. In the first group, we want to place the data types starting from the most generated and mostly used ones. Cost function as a whole is not considered in this group, and Algorithm 1 and Algorithm 2 belong to this group. We consider the cost function as whole in the second group. The algorithms in this group not only consider data generation volumes, but also the data access patterns of the applications. Algorithms 3 and 4 are the elements of this group, and as it will be clear in the simulation results given in Chapter 7 that they converge best to the output of linear solvers.

5.1 Algorithm 1

Algorithm 1: Placement of mostly generated data near to the mostly used FCU.

Data: M, N, D, A, L, AR, AF, GV, SC
Result: P
1 $DG \leftarrow$ **matrixColumnSum**(GV)
2 **mergeSortDescendingWIndices**(DG,0,D-1)
3 $dataAccess \leftarrow$ **matrixMultiply**(AR, AF)
4 $P \leftarrow$ **placeDataTypeAlg1**$(M, N, D, L, SC, DG, dataAccess, dataUse)$
5 **return** P

In Algorithm 1, we try to place the mostly generated data type to the nearest data center, where it is mostly used. This algorithm starts from placing the mostly generated data types to the least, and if the nearest data center is full, which can be the case if the nearest data center is a FC, it is placed to the second best data center and so on.

If we elaborate the algorithm, from line 1 to line 2, data generation volumes of each data type is calculated and sorted from the mostly generated to the least. It will take $\mathbf{O}(D \times N)$ time, since the dominating part in the section is matrix column sum operation. It requires summing all the elements in **GV** matrix which takes $\mathbf{O}(D \times N)$. Sorting of the total data generation volumes does not take that much time because in most of the architectures logarithm of the total number of data generated $\mathbf{log}(D)$ is less than the total number of FCUs **N**. Line 3 is a

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

```
1  Initialize P matrix: P ← 0
2  Initialize UC vector: UC ← 0
3  for i ← 0 to D − 1 do
4  |   tempMax ← 0
5  |   maxInd ← 0
6  |   for j ← 0 to N − 1 do
7  |   |   if dataAccess[j][DG[0][i]] > tempMax then
8  |   |   |   tempMax ← dataAccess[j][DG[0][i]]
9  |   |   |   maxInd ← j
10 |   minDist ← ∞
11 |   minInd ← 0
12 |   for j ← 0 to M + N − 1 do
13 |   |   if L[M + maxInd][j] < minDist then
14 |   |   |   if j < M then
15 |   |   |   |   minDist ← L[M + maxInd][j]
16 |   |   |   |   minInd ← j
17 |   |   |   else if UC[j − M] + DG[1][i] ≤ SC[j − M] then
18 |   |   |   |   minDist ← L[M + maxInd][j]
19 |   |   |   |   minInd ← j
20 |   if minInd ≥ M then
21 |   |   UC[minInd − M] ← UC[minInd − M] + DG[1][i]
22 |   P[i][minInd] ← 1
23 return P
```

matrix multiplication and will take $\mathbf{O}(N \times A \times D)$ without using an optimized matrix multiplication algorithm. This matrix multiplication is mandatory for calculating the data access patterns of each FCU to the corresponding data type. Then the placement procedure starts in line 4, and takes $\mathbf{O}(D \times (M + N))$.

In the placement procedure, from line 1 to line 2 the initialization of data placement matrix and used capacities are set. Starting from line 3, the mostly generated data type is placed to the nearest data center available where it is mostly used.

5.2 Algorithm 2

Algorithm 2: 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 DG ← matrixColumnSum(GV)
2 dataAccess ← matrixMultiply(AR, AF)
3 tempAccess ← matrixColumnSum(dataAccess)
4 for i ← 0 to D − 1 do
5 |   dataUse[0][i] ← i
6 |   dataUse[1][i] ← tempAccess[i] × DG[i]
7 mergeSortDescendingWIndices(dataUse,0,D-1)
8 P ← placeDataTypeAlg2(M, N, D, L, SC, DG, dataAccess, dataUse)
9 return P
```

Algorithm 2 is a slightly different version of Algorithm 1. It starts placing from the mostly accessed data types instead of the mostly generated ones to the nearest data centers, where they are mostly used without considering application running profiles.

In both algorithms, DP agent can monitor application run profiles, and data access patterns, while DC agent can give the statistics of data generation volumes. According to the outputs of DC and DP agents, Algorithm 2 sorts data types depending on their total access volumes, and this is the main difference between Algorithm 1 and Algorithm 2. Then, placement starts from the most accessed data type. The rule is to find for each data type a suitable data center (an FCU) which preferably the nearest to 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 the data is placed. If the available capacities of the nearby FCUs are not enough for storing, the data type is placed to the nearest CC. The existence of CCs guarantees the convergence of both algorithms, since they have unlimited storage capacities.

In pseudo-code of the algorithm, from line 2 to 7, data access volumes for each

Procedure placeDataTypeAlg2(M, N, D, L, SC, DG, dataAccess, dataUse)

```
1  Initialize P matrix: P ← 0
2  Initialize UC vector: UC ← 0
3  for i ← 0 to D − 1 do
4  |   tempMax ← 0
5  |   maxInd ← 0
6  |   for j ← 0 to N − 1 do
7  |   |   if dataAccess[j][dataUse[0][i]] > tempMax then
8  |   |   |   tempMax ← dataAccess[j][dataUse[0][i]]
9  |   |   |   maxInd ← j
10 |   minDist ← ∞
11 |   minInd ← 0
12 |   for j ← 0 to M + N − 1 do
13 |   |   if L[M + maxInd][j] < minDist then
14 |   |   |   if j < M then
15 |   |   |   |   minDist ← L[M + maxInd][j]
16 |   |   |   |   minInd ← j
17 |   |   |   else if UC[j − M] + DG[i] ≤ SC[j − M] then
18 |   |   |   |   minDist ← L[M + maxInd][j]
19 |   |   |   |   minInd ← j
20 |   if minInd ≥ M then
21 |   |   UC[minInd − M] ← UC[minInd − M] + DG[i]
22 |   P[i][minInd] ← 1
23 return P
```

data type are calculated. It is an indication of how often each data type is used by all FCUs. After the calculation, they are sorted from the most accessed to the least. This process takes $\mathbf{O}(N \times A \times D)$ time without using any kind of matrix multiplication optimization. Run times of both Algorithm 1 and 2 are dominated by the calculation of data access patterns 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 2 also takes $\mathbf{O}(D \times (M + N))$.

In the placement procedure, from line 1 to line 2 the initial values of data placement matrix and used capacities are set. Starting from line 3, the most accessed data type is placed to an available data center which is nearest to the FCU 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.

Both placement procedures for the first group of algorithms are almost same except the decision start matrices **DG** and **dataUse**. For achieving better performance in the first group of algorithms the bottleneck should be improved, which is a rectangular matrix-matrix multiplication in both.

5.3 Algorithm 3

Algorithm 3: Placement of data that affect cost function most to the nearest data center where it is used most one-by-one.

Data: M, N, D, A, L, AR, AF, GV
Result: P

$DG \leftarrow$ **matrixColumnSum**(GV)
mergeSortDescendingWIndices(DG,0,D-1)
$normDenom \leftarrow \sum_{i=0}^{N-1} \sum_{j=0}^{A-1} ar_{i,j}$
$NAR \leftarrow AR/normDenom$
$rowScaling \leftarrow$ **matrixRowSum**(**sgn**(AF))
for $i \leftarrow 0$ **to** $A-1$ **do**
    for $j \leftarrow 0$ **to** $D-1$ **do**
        $SFD[i][j] \leftarrow$ **sgn**$(AF[i][j])/rowScaling[i]$
$DCF \leftarrow$ **matrixMultiply**(NAR, SFD)
$P \leftarrow$ **placeDataTypeAlg3**$(M, N, D, L, SC, DG, DCF)$
return P

The idea behind the second group of algorithms is to find a suitable place for data types starting from the most effective one on the defined average latency function. In this group, efficiencies of each data type are calculated according to the variables defined in Chapter 4, and then placement choices for each data type are ordered for minimizing the average latency. Since the problem is a linear optimization 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 optimal solution. So we try to find a suitable place for data types starting from the most effective

Procedure placeDataTypeAlg3(M, N, D, L, SC, DG, DCF)

```
1  Initialize P matrix: P ← 0
2  Initialize UC vector: UC ← 0
3  for i ← 0 to D − 1 do
4  |  for j ← 0 to M + N − 1 do
5  |  |  pL[j] ← 0
6  |  |  for k ← M to M + N − 1 do
7  |  |  |  pL[j] ← pL[j] + (L[j][k] × DCF[k − M][DG[0][i]])
8  |  minDist ← ∞
9  |  minInd ← 0
10 |  for j ← 0 to M + N − 1 do
11 |  |  if pL[j] < minDist then
12 |  |  |  if j < M then
13 |  |  |  |  minDist ← pL[j]
14 |  |  |  |  minInd ← j
15 |  |  |  else if UC[j − M] + DG[i] ≤ SC[j − M] then
16 |  |  |  |  minDist ← pL[j]
17 |  |  |  |  minInd ← j
18 |  if minInd ≥ M then
19 |  |  UC[minInd − M] ← UC[minInd − M] + DG[1][i]
20 |  P[i][mindInd] ← 1
21 return P
```

one, and Algorithm 3 is the first of this group. It considers both the normalized applications running frequencies and data requirement patterns of the application. This distinguishes both Algorithm 3 and Algorithm 4 from the first group. Data requirement pattern is different than the data access one, since whenever an application needs a certain data type, it definitely has to access it. Hence, the matrix of data access frequencies of the applications denoted by **AF** is changed to a binary matrix while calculating the data requirement pattern. To be more precise, if an application's access frequency to a data type is greater than 0, then the data requirement of the application to that type of data is '1', this is also the case in formal problem formulation given in equation 4.4; otherwise it is '0'.

There are two important issues in this algorithm. First one, each data type is given precedence according to total generation volume. For example, if data type **A**is generated more than data type **B**, the precedence of data type **A** would

be higher than data type **B**. The second issue is that each data type is placed according to its precedence. If we consider the previous example, data type A would be placed before than data type.

Algorithm 3 starts from calculating which data types are generated mostly in given network, from line 1 to 2; it again takes $\mathbf{O}(D \times N)$ according to the previous assumption explained in Section 5.1. After sorting which data types are generated mostly, normalization of the applications running frequencies and data dependency matrices are calculated from line 3 to 8. Normalization of the applications running frequencies takes $\mathbf{O}(N \times A)$ and changing data access frequency matrix (**AF**) to binary takes $\mathbf{O}(A \times D)$ time. After obtaining normalized applications running frequency and data requirement matrices; we multiply them with each other to get which data types are required in each FCU (line 9). If no optimized matrix multiplication algorithm is used, it takes $\mathbf{O}(N \times A \times D)$. Then, the placement procedure starts.

From line 1 to 2 of placement procedure, initialization of the data placement matrix and used capacity vector variables are set. It takes $\mathbf{O}(D \times (M + N))$, which is same for all algorithms. Starting from line 3, algorithm starts placing from mostly generated data type by calculating total latency of each data center to others. This gives the idea of how the cost function is affected if data is placed that data center (from line 4 to line 7). After the calculation of how the cost function is affected, placement of data types start. The goal is to place all data types to their corresponding best places to minimize the cost function. Starting from the mostly generated data type, all placement choices of each data type are checked and the data is placed to the best among the available ones. Then, the second mostly generated data type is placed and so on. The placement procedure takes $\mathbf{O}(D \times (M + N)^2)$.

5.4 Algorithm 4

Algorithm 4: 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

$DG \leftarrow$ **matrixColumnSum**(GV)
mergeSortDescendingWIndices(DG,0,D-1)
$normDenom \leftarrow \sum_{i=0}^{N-1} \sum_{j=0}^{A-1} ar_{i,j}$
$NAR \leftarrow AR/normDenom$
$rowScaling \leftarrow$ **matrixRowSum**(**sgn**(AF))
for $i \leftarrow 0$ **to** $A-1$ **do**
    for $j \leftarrow 0$ **to** $D-1$ **do**
        $SFD[i][j] \leftarrow$ **sgn**$(AF[i][j])/rowScaling[i]$
$DCF \leftarrow$ **matrixMultiply**(NAR, SFD)
$P \leftarrow$ **placeDataTypeAlg4**$(M, N, D, L, SC, DG, DCF)$
return P

As in the case of Algorithm 3, Algorithm 4 also tries to minimize the defined cost function considering as a whole, and converge to the linear solution, but there exists a small difference in the placement procedure. Algorithm 4 does not guarantee that each data type is placed according to their total data generation volume precedence, which is one of the rules used in the previous algorithm. In the placement procedure of this algorithm mentioned rule is broken, and the available best choices of each data type is assumed to be critical. They are placed whether their best places are available or not. This makes Algorithm 4 is a tokenized algorithm. This is the major difference of this algorithm and the others. In the first and second algorithms, data types mostly generated and accessed ones are placed to the closest data centers respectively, where they are used most; in the third algorithm, starting from the mostly generated data type placement is done according to where the cost function is minimized. However, in the last algorithm starting from the mostly generated data type, every data type is placed to their best choices.

To explain the placement procedure of this algorithm deeply, there are rounds, and for each round according to the sorted remaining choices, data is placed if

```
Procedure placeDataTypeAlg4(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 − 1 do
8          if dataPlaced[DG[0][i]] ≠ 1 then
9              for j ← 0 to M + N − 1 do
10                 pL[0][j] ← j
11                 pL[1][j] ← 0
12                 for k ← M to M + N − 1 do
13                     pL[1][j] ← pL[1][j] + (L[j][k] × DCF[k − M][DG[0][i]])
14             mergeSortAscendingWIndices(pL,0,M+N-1)
15             if pL[0][turn] ≥ M then
16                 if UC[pL[0][turn] − M] + DG[1][i] ≤ SC[pL[0][turn] − M]
                   then
17                     P[DG[0][i]][pL[0][turn]] ← 1
18                     UC[pL[0][turn] − M] ← UC[pL[0][turn] − M] + DG[1][i]
19                     dataPlaced[DG[0][i]] ← 1
20                     placedDataCnt ← placedDataCnt + 1
21             else
22                 P[DG[0][i]][pL[0][turn]] ← 1
23                 dataPlaced[DG[0][i]] ← 1
24                 placedDataCnt ← placedDataCnt + 1
25     turn ← turn + 1
26 return P
```

the capacity of the remaining best choice is available. This seems complicated, but it will become clear by the following example. Consider three data types to be placed in two FCUs and a CC. Starting from the 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 the 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 directly placed in CC. No other data types are left except data type 2, and the first placement turn is

over since all the best choices for each data type are visited. 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, the following best choice is taken into account, and assume that the second best choice of data type 2 is FCU 2; but again its capacity is not big enough for storing the data type 2, then the 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.

All of the discussion about the algorithm part of Algorithm 3 is also valid for Algorithm 4. The difference starts with the initialization part of the placement procedure. A new vector called **dataPlaced** is defined in Algorithm 4, and it is a flag indicating whether the corresponding data type is placed or not. Since the last algorithm does not guarantee to place data types in order according to their data generation volume precedence, we have to keep track of which data types are placed and which are not. Except the flag, there also exists two other variables: **placedDataCnt** and **turn**. The first of them is used for checking whether all data types are placed or not, and the second one is used for in which round the placement algorithm is.

In the placement procedure, from line 1 to line 3 initialization of the vectors is completed. Lines 4 and 5 define the required variables for tokens. Starting from line 6, algorithm places each data type to its best choice turn-by-turn. Algorithm terminates when all data types are placed, and it is checked by **placedDataCnt**. Run time of the algorithm is not different than the previous one with wise implementation. In the pseudo-code it seems $\mathbf{O}(D^2 \times (M+N)^2)$, but we do not need to calculate the effects of placing each data type to all data centers again and again in all rounds. It is enough to calculate it once, and this makes the run time of the algorithm $\mathbf{O}((D \times (M+N)^2) + (D^2 \times (M+N)))$. Since most of the time the relation $\mathbf{D} \leq (\mathbf{M}+\mathbf{N})$ holds, the run time is same as Algorithm 3: $\mathbf{O}(D \times (M+N)^2)$.

The major advantage of using all proposed algorithms is their parallelizable nature, since they mostly consist of vector and matrix operations. This parallelization potential of the algorithms makes them suitable for running on multi-core architectures efficiently.

5.5 Summary

In this chapter, we present four heuristic algorithms for the data placement problem, which can be used instead of the linear model given in Chapter 4. The problem with linear model is the run-time since the solution space grows exponentially as the number of data centers, data types and running applications increase. On the contrary, our proposed algorithms have cubic run-times at most.

All algorithms assume that outputs of DC and DP agents, which are data generation volumes, data access patterns and running frequencies of the applications, are known a priori. By using these outputs, proposed algorithms try to place the data types efficiently.

Proposed algorithms can be grouped in two: the first group consists of the algorithms focusing on data patterns and the second one considers the average latency function as a whole.

All of the proposed algorithms can easily run in any of the available cloud data centers in the network, while they obtain outputs of the agents. This can be achieved by using a centralized mechanism, such as software-defined networking (SDN) controller.

Chapter 6

Hybrid Fog-Cloud Computing Based Network Topology Modeling

Until now, we have presented the hybrid data placement structure and the algorithms. As mentioned in previous chapters of this thesis, most of the work about the IoT networks in a large area, such as smart city scale, is hypothetical. There exists no well-known deployed architecture, regarding that fact to verify our proposed algorithms and network structure, we need to generate a simulation environment with the elements of our proposed architecture: IoT nodes, fog and cloud computing data centers.

In the generation of sample network topology, we consider IoT nodes and fog computing nodes together forming the architectural element, which we propose, the Fog Computing Units (FCUs). We also bear in mind the neighborhood concept in the smart cities, and we divide the city in smaller regions to form these neighborhoods. In each neighborhood, we place the IoT nodes and fog computing data centers together, and name each neighborhood as an FCU. After constituting the neighborhoods, in other words FCUs, we have cloud computing data centers to be placed, and we place them to the intersection points of the neighborhoods.

Under the light of these assumptions, we form the smart city network in a rectangular region and divide this large area into smaller ones calling each of them as neighborhoods. The corners of these smaller rectangles are the possible candidate places for CCs, and during the placement procedure of CCs, we consider the load balancing between them. Finally, we form the sample network topology to verify our proposed algorithms.

In the next sections of this chapter, we present algorithms used in our hybrid network topology generation. Section 6.1 describes the algorithms for generating the smart city network with neighborhoods, and Section 6.2 gives some experimental results related to the algorithms used from different perspectives, such as the balance between rectangles, the workloads of CCs while forming CC and FCU clusters which may be used for other services.

6.1 Hybrid Network Topology Modeling Algorithm

Algorithm for generating sample network topology starts by creating neighborhoods in a rectangular city area. After generating FCUs, we get the candidate places for CCs, and among these we choose the suitable ones to place CCs according to some criteria. Then, we obtain the sample network topology to verify our proposed algorithms.

In this section we want to explain the algorithms for obtaining FCUs and choosing CC places in a smart city model. Section 6.1.1 explains how we divide the big city rectangle into smaller ones to get neighborhoods, and Section 6.1.2 explains how we choose the CC places and form the FCU connections in between.

6.1.1 Rectangular Area Creation

The main structure of the proposed architecture is the fog computing units (FCUs). To model FCUs, we need to generate some areas representing these elements and to achieve this, we model the city as a rectangular area mentioned in previous chapters, and divide it into smaller rectangles. Each rectangle is assumed to be a neighborhood in the city. Inside these rectangles, IoT nodes and fog computing data centers reside. As mentioned in Section 3.1, every IoT node is connected to a FC in the neighborhood, and IoT nodes are distributed randomly in each of these rectangular areas while there exists FC data center(s) serving them. For the sake of concentrating our goal of verifying our proposed algorithms and the network architecture, we do not mainly focus on what is happening inside FCUs, such as how the internal connections are formed, which data centers are chosen by which IoT nodes, etc. and leave it as a future work. But what we assume is the following, nodes constituting the FCUs are uniformly distributed along the region.

We start the algorithm by getting X & Y lengths of the large rectangle, the total number of rectangles to be generated inside, and limits of how many times X and Y axes to be divided. After obtaining the size and total number of rectangles to be generated, we start dividing the big rectangle horizontally. We obtain horizontal rows, and when we divide each of these rows vertically, smaller rectangular regions appear. The algorithm terminates when the required number of rectangles are created. In each horizontal row, the starting Y coordinate of each rectangle is equal, but their X coordinates are different. The output of the algorithm gives us random rectangles in quadruples: two start coordinates of each rectangle (X & Y axes) and lengths of each sides (X & Y sides).

6.1.1.1 Horizontal and Vertical Division Counts

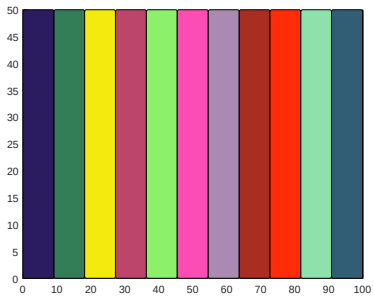
In the division process of the big rectangle (city) to the smaller ones (neighborhoods), we first focus on how we obtain smaller rectangles inside the big one. And

Algorithm 5: Calculation of horizontal and vertical cut counts.

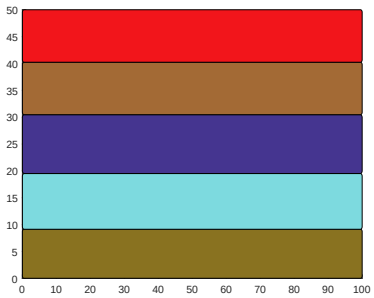
Data: Max_X_{div}, Max_Y_{div}, N
Result: hRow_Cnt & vCut_Cnt_Each_Row

```
1  hCutInd ← 1
2  rectReq ← N
3  while rectReq > 0 do
4  |   checked ← 0
5  |   while checked = 0 do
6  |   |   vCutCnt ← random_integer(1, Max_X_div)
7  |   |   if (vCutCnt + (Max_X_div * (Max_Y_div − hCutInd))) ≥ rectReq
   |   |   then
8  |   |   |   if vCutCnt > rectReq then
9  |   |   |   |   vCutCnt ← rectReq
10 |   |   |   vCut_Cnt_Each_Row[hCutInd − 1] ← vCutCnt
11 |   |   |   hCutInd ← hCutInd + 1
12 |   |   |   rectReq ← rectReq − tempXDivCnt
13 |   |   |   checked ← 1;
14 hRow_Cnt ← hCutInd − 1
15 return hRow_Cnt & vCut_Cnt_Each_Row
```

we decide to divide the big rectangle in one direction first: either horizontally or vertically, then divide it in the other direction instead of generating arbitrary random rectangles (see Figure 6.1). Bear in mind that, we also do not want to use a grid-like rectangular areas, since the total number of required rectangles may not always satisfy the product of vertical and horizontal cuts, and it significantly degrades randomness of the areas of the generated rectangles.



(a) An example of 12 vertical rectangle cuts.



(b) An Example of 5 horizontal rectangle cuts.

Figure 6.1: Examples of vertical and horizontal rectangle cuts.

In the beginning of the algorithm, we do not know exactly how many horizontal divisions and how many vertical cuts needed, but what we do know is the maximum number of total horizontal cuts (Max_Y_{div}) and the maximum allowed number of vertical cuts (Max_X_{div}) in each horizontal segment; so we start by considering each horizontal slice one-by-one (see Figure 6.3).

For each horizontal slice, we generate a random integer to find out how many vertical cuts needed for that segment, then we check whether the remaining number of rectangles can be satisfied by the other horizontal cuts if they are all divided with maximum allowable vertical cuts. Pseudo-code of the explained algorithm is given in Algorithm 5.

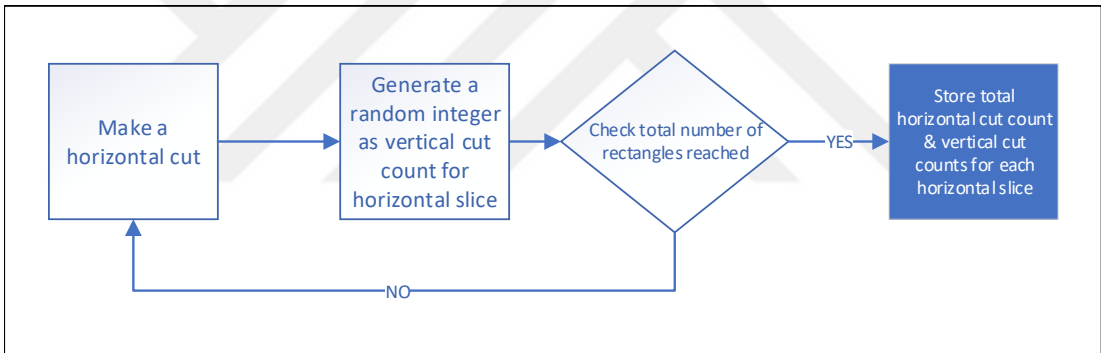


Figure 6.2: Horizontal & vertical cut flowchart.

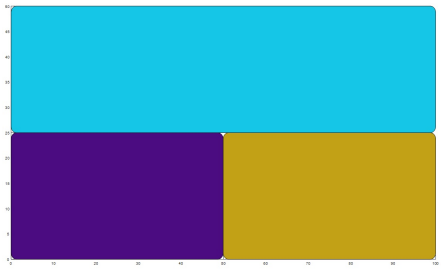


Figure 6.3: An example of vertical cuts over horizontal ones.

6.1.1.2 Choosing Y-Points

In Chapter 6.1.1.1, we calculate how many horizontal cuts needed in total and vertical cuts required for each horizontal slice. After that, we need to find the

coordinates of each horizontal cut, in other words figure out the Y-axis points of each rectangle. For each Y-axis point, we randomly select a point from uniform distribution. The critical point is if all points are chosen totally random in the whole grid, imbalance between rectangles increases. We overcome this by defining minimum and maximum limits according to the horizontal cuts for each random variable. The pseudo-code of this process is given in Algorithm 6.

Algorithm 6: Choosing Y-points.

Data: hRow_Cnt, Y, yDiv_Ratio
Result: yAxis_Pts

$yAxis\_Pts[hRow\_Cnt] \leftarrow Y$
$yAxis\_Pts[0] \leftarrow 0$
$leftY \leftarrow Y$
for $i \leftarrow 0$ **to** $hRow\_Cnt - 1$ **do**
    $tempPartSize \leftarrow hRow\_Cnt - i$
    $minL \leftarrow (leftY/tempPartSize) * (1 - yDiv\_Ratio)$
    $maxL \leftarrow (leftY/tempPartSize) * (1 + yDiv\_Ratio)$
    $chosenL \leftarrow minL + ((maxL - minL) * random(0, 1))$
    $leftY \leftarrow leftY - chosenL$
    $yAxis\_Pts[i] \leftarrow yAxis\_Pts[i - 1] + chosenL$
return $yAxis\_Pts$

The limit of each random Y-axis point is input to the algorithm ($yDiv\_Ratio$), and each iteration remaining Y-axis is divided by left horizontal cut count. The algorithm starts considering the whole Y-axis (large rectangle's Y-side), and after choosing one random point from U˜$[(1 - yDiv\_Ratio) * (Y/H_{cut\_count}), (1 + yDiv\_Ratio) * (Y/H_{cut\_count})]$, left Y-length is divided by one minus of the total horizontal cut count, and the algorithm terminates after finding the last horizontal slice's Y-axis point.

In each iteration one point is chosen which is perturbed from the expected mean of the remaining Y length with a variance related to the input $yDiv\_Ratio$. We expect the distribution of points is correlated with the Central Limit Theorem (CLT), and we show this with example distributions in Chapter 6.2.1.

Algorithm 7: Choosing X-points.

Data: hRow_Cnt, vCut_Cnt_Each_Row, X, Y, xDiv_Ratio
Result: xAxis_Pts

$tempInd \leftarrow 0$
$xAxis\_Pts[0] \leftarrow 0$
for $i \leftarrow 0$ **to** $hRow\_Cnt - 1$ **do**
    $tempInd \leftarrow \textbf{sum}(vCut\_Cnt\_Each\_Row[0 \rightarrow i]) + i$
    $xAxis\_Pts[tempInd] \leftarrow X$
    $leftX \leftarrow X$
    for $j \leftarrow 0$ **to** $vCut\_Cnt\_Each\_Row[i] - 1$ **do**
        $tempPartSize \leftarrow vCut\_Cnt\_Each\_Row[i] - j$
        $minL \leftarrow (leftX/tempPartSize) * (1 - xDiv\_Ratio)$
        $maxL \leftarrow (leftX/tempPartSize) * (1 + xDiv\_Ratio)$
        $chosenL \leftarrow minL + ((maxL - minL) * random(0, 1))$
        $leftX \leftarrow leftX - chosenL$
        $tempInd \leftarrow tempInd + 1$
        $xAxis\_Pts[tempInd] \leftarrow xAxis\_Pts[tempInd - 1] + chosenL$
    $tempInd \leftarrow tempInd + 2$
return $xAxis\_Pts$

6.1.1.3 Choosing X-Points

In the previous Chapter 6.1.1.2, we decide how to choose Y-axis points of each horizontal cut. What is left before creating random number of rectangles is the vertical cuts on the horizontal ones, in other words the X-axis points of the rectangles.

Bearing in mind the balance between rectangles, for each horizontal cut we divide the X-axis into regions and perturb the division points with some limits, which is an input to the algorithm ($xDiv\_Ratio$). This is done as in the case of choosing Y-axis points, we first decide the expected mean which is the mid-point of the perturbation and within the limits we generate a random number using uniform distribution. Details, example distributions and extra explanation are given in the experimental results section (Chapter 6.2.1).

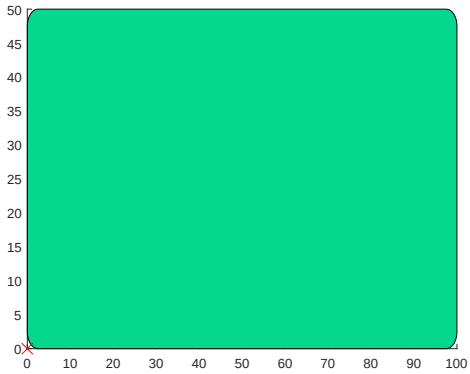


Figure 6.4: An example of a rectangle generated by the algorithm.

6.1.1.4 Creating Rectangles in Quadruples

In the previous sections, we have marked the X and Y points of the each cut, and this section we define the rectangles in quadruples: X & Y coordinates of the left-bottom point and the side lengths of the rectangle. An example run of the algorithm is given in Figure 6.4. The left-bottom point of the rectangle is marked by red 'X' in the example. Quadruple notation of the generated rectangle is '{0, 0, 100, 50}', meaning that the coordinates of the left-bottom point is (0, 0) and the X & Y side-lengths are 100 and 50 respectively.

Pseudo-code of the generating rectangle algorithm is given in Algorithm 8. Other than creating the rectangle in quadruple, it calculates some critical values for the network generation such as areas and centers of the generated rectangles and inner intersection points of each rectangle.

The importance of these points will become clear in the next sections, but we can explain briefly as the following. Since at the beginning, we assume that IoT nodes are uniformly distributed along neighborhoods, so the area is used when calculating the traffic related data in the corresponding neighborhood, and it is critical while making a decision of which cloud computing data center is better for the corresponding FCU according to some balance criteria. Rectangle centers are used for calculating distances between FCUs and CCs, and the inner intersection points are the candidate places of CCs.

6.1.2 Placement of CCs and Assigning FCUs

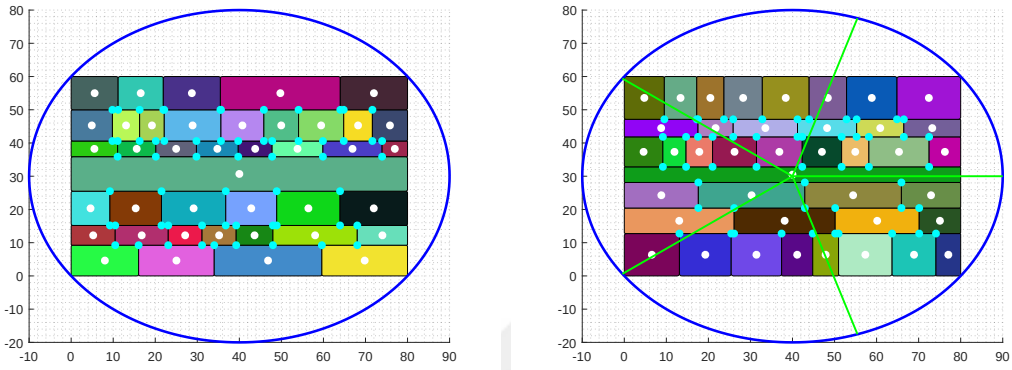
In the previous sections, we have generated rectangles for denoting FCUs. In this section, we want to place the remaining critical element of the proposed architecture, which is cloud computing data centers (CCs). As described in Section 3.1, CCs are the ultimate places to store data when the available storage capacity of FCUs does not satisfy the demand. As a consequence of that, for easily serve to the FCUs we need to place CCs near to more than one FCU. Under the light of this goal, we choose placing CCs in the inner corners, i.e. the intersection points, of the FCUs.

The algorithm used for choosing the places of CC points is k-means clustering. The difference is in the first iteration, instead of randomly chosen cluster centers as in the case of k-means, we choose the cluster centers according to projection of a circle to the big city rectangle. We draw a circumscribed circle of the big rectangle. Afterwards, we divide it into slices according to the total number of clusters to be generated. Finally in the first iteration, we choose the crossing points of circle radii with big rectangle as the first center points of the clusters (see Figure 6.6). Initial cluster center choosing algorithm is given in Algorithm 9.

As mentioned previously, load balancing between CCs should be kept in mind. We use two different clustering strategies: first one which does not consider load balancing is try to connect FCUs to the nearest CC and the second one is for load balancing which chooses the less crowded CC if possible.

In the first algorithm, whose pseudo-code is given in Algorithm 10, we connect FCUs to the nearest CC. All FCUs have equal weight, so the size of rectangles or the number of IoT nodes in each FCU is not considered. Although we use mesh logical topology, there may exist some services, except data sharing, available for FCUs which may require star topology and delay is important. Regarding that by choosing the nearest CC in the first strategy supports this demand.

To explain the pseudo-code, we define maximum iteration count to block the oscillations and guarantee the termination of algorithm. In each iteration the



(a) An example of a circumscribed circle.

(b) An example of a divided circumscribed circle.

Figure 6.5: Examples of undivided/divided circumscribed circles.

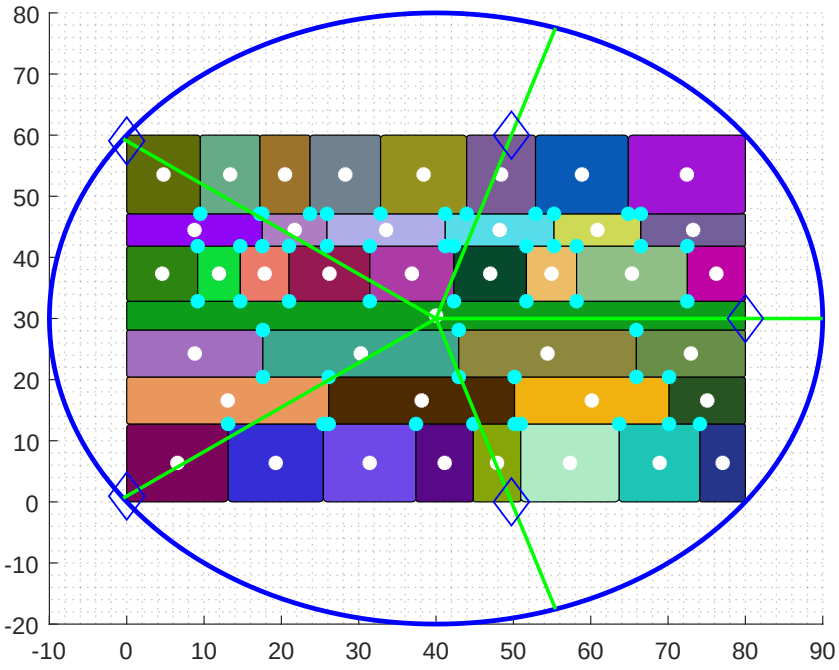


Figure 6.6: Cluster centers in the first iteration of k-means algorithm.

distance of each FCU center to the given cluster is calculated, and if there exists a nearer one, we move the FCU to that cluster. At the end of each iteration, new cluster center is calculated, and if no FCUs move or the maximum number of iterations reached, algorithm terminates.

In the second strategy (Algorithm 11), used for load balancing, we update the cluster centers of k-means algorithm with bearing in mind the areas of the connected FCUs to that CC. Also there exists a penalization mechanism when calculating distance to cluster centers. If there exists crowded FCUs connected to the corresponding CC, distance from FCU to that CC becomes large according to the formula given below.

$$Distance = \frac{Total\_Cluster\_Area + FCU\_Area}{FCU\_Area} \times \sqrt{(X_{cc} - X_{fcu})^2 + (Y_{cc} - Y_{fcu})^2} \tag{6.1}$$

Equation 6.1 gives priority to less crowded CCs to be connected by the FCU. The rest of the algorithm is same as the first one except the cluster center calculation. We update them at the end of each iteration according to the areas of FCUs connected to CC.

After running k-means algorithm with different strategies, we obtain the center points of each cluster with FCUs assigned, and we choose the nearest intersection point of rectangles as the place for the corresponding CC. We connect all FCUs to the placed CC, finally we obtain our network topology with CCs placed in the intersection point of the FCUs, and each FCU is modeled as a rectangle. Examples of these two strategies are given Figures 6.7& 6.8.

We want to emphasize that the clustering may become an important issue in a star topology network, and this may be the case in each FCU. But for the verification purpose of the proposed algorithms and model, we use mesh logical topology.

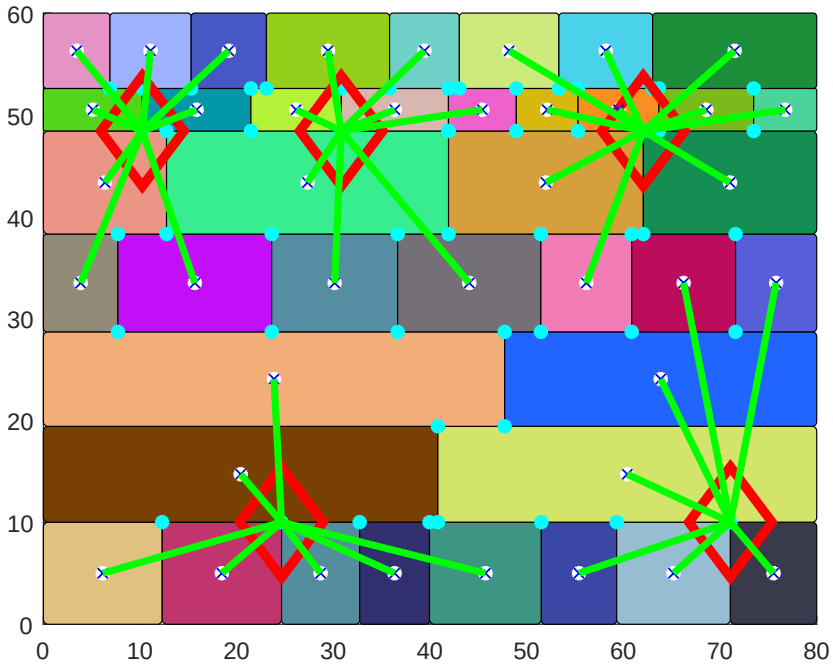


Figure 6.7: An example of k-means choosing nearest CC.

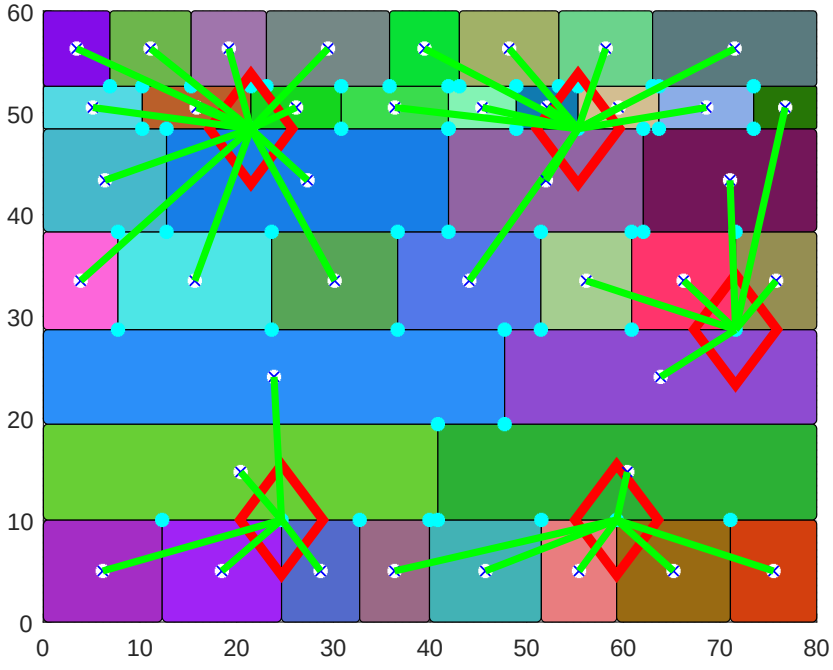


Figure 6.8: An example of k-means choosing less crowded CC.

Algorithm 8: Creating rectangles.

Data: N, X, Y, hRow_Cnt, xAxis_Pts, yAxis_Pts
Result: createdRects, rectAreas, rectCenters, innerCorners

```
totalXPts ← N + hRow_Cnt
yPtInd ← 0
cornerInd ← 0
rectInd ← 0
for i ← 0 to totalXPts − 2 do
    if xAxis_Pts[i] = X then
        yPtInd ← yPtInd + 1
    else
        xLength ← xAxis_Pts[i + 1] − xAxis_Pts[i]
        yLength ← yAxis_Pts[yPtInd + 1] − yAxis_Pts[yPtInd]
        createdRects[rectInd][0] ← xAxis_Pts[i]
        createdRects[rectInd][1] ← yAxis_Pts[yPtInd]
        createdRects[rectInd][2] ← xLength
        createdRects[rectInd][3] ← yLength
        rectAreas[rectInd] ← xLength ∗ yLength
        rectCenters[rectInd][0] ← xAxis_Pts[i] + xLength/2
        rectCenters[rectInd][1] ← yAxis_Pts[i] + yLength/2
        rectInd ← rectInd + 1
    if xAxis_Pts[i] ≠ 0 then
        if yAxis_Pts[yPtInd] ≠ 0 then
            innerCorners[cornerInd][0] ← xAxis_Pts[i]
            innerCorners[cornerInd][1] ← yAxis_Pts[yPtInd]
            cornerInd ← cornerInd + 1
        if yAxis_Pts[yPtInd] + yLength < Y then
            innerCorners[cornerInd][0] ← xAxis_Pts[i]
            innerCorners[cornerInd][1] ← yAxis_Pts[yPtInd] + yLength
            cornerInd ← cornerInd + 1
return createdRects, rectAreas, rectCenters, innerCorners
```

Algorithm 9: Setting initial cluster centers.

Data: M, X, Y
Result: CC_Coords

$tempCriticalAng \leftarrow arctan(Y/X)$
$angLimits \leftarrow \{tempCriticalAng, \pi/2, \pi - tempCriticalAng, \pi, \pi + tempCriticalAng, 3\pi/2, 2\pi - tempCriticalAng, 2\pi, 2\pi + tempCriticalAng\}$
$tempAngLimCheck \leftarrow 0$
for $i \leftarrow 0$ **to** $M - 1$ **do**
    $angOfPt \leftarrow i * (M/2\pi)$
    while $angOfPt > angLimits[tempAngLimCheck]$ **do**
        $tempAngLimCheck \leftarrow tempAngLimCheck + 1$
    switch *tempAngLimCheck* **do**
        case *0* **do**
            $CC\_Coords[i][0] \leftarrow X$
            $CC\_Coords[i][1] \leftarrow (Y/2) + ((X/2) * tan(angOfPt))$
        case *1* **do**
            $CC\_Coords[i][0] \leftarrow (X/2) + ((Y/2) * cot(angOfPt))$
            $CC\_Coords[i][1] \leftarrow Y$
        case *2* **do**
            $CC\_Coords[i][0] \leftarrow (X/2) - ((Y/2) * cot(\pi - angOfPt))$
            $CC\_Coords[i][1] \leftarrow Y$
        case *3* **do**
            $CC\_Coords[i][0] \leftarrow 0$
            $CC\_Coords[i][1] \leftarrow (Y/2) + ((X/2) * tan(\pi - angOfPt))$
        case *4* **do**
            $CC\_Coords[i][0] \leftarrow 0$
            $CC\_Coords[i][1] \leftarrow (Y/2) - ((X/2) * tan(angOfPt - \pi))$
        case *5* **do**
            $CC\_Coords[i][0] \leftarrow (X/2) - ((Y/2) * cot(angOfPt - \pi))$
            $CC\_Coords[i][1] \leftarrow 0$
        case *6* **do**
            $CC\_Coords[i][0] \leftarrow (X/2) + ((Y/2) * tan(angOfPt - 3\pi/2))$
            $CC\_Coords[i][1] \leftarrow 0$
        case *7* **do**
            $CC\_Coords[i][0] \leftarrow X$
            $CC\_Coords[i][1] \leftarrow (Y/2) - ((X/2) * cot(angOfPt - 3\pi/2))$
return *CC_Coords*

Algorithm 10: Choose CC places & assign FCUs to the nearest CC.

Data: M, N, innerCorners, rectCenters, rectAreas, X, Y, maxIter
Result: CC_Coords, FCU_Clust

```
movedFCUCnt ← 1
iterCnt ← 0
for i ← 0 to N − 1 do
    oldClust[i] ← 0
    FCU_Clust[i] ← 1
for i ← 0 to M − 1 do
    clustSize[i] ← 0
while (movedFCUCnt > 0) & (iterCnt < maxIter) do
    movedFCUCnt ← 0
    for i ← 0 to N − 1 do
        oldClusters[i] ← FCU_Clust[i]
        minDist ← ∞
        minInd ← 0
        for j ← 0 to M − 1 do
            tempDist ← sqrt((CC_Coords[j][0] − rectCenters[i][0])^2 +
            (CC_Coords[j][1] − rectCenters[i][1])^2)
            if tempDist < minDist then
                minDist ← tempDist
                minInd ← j
        if iterCnt ≠ 0 then
            clustSize[oldClust[i]] ← clustSize[oldClust[i]] − 1
        FCU_Clust[i] ← minInd
        clustSize[minInd] ← clustSize[minInd] + 1
        if oldClust[i] ≠ FCU_Clust[i] then
            movedFCUCnt ← movedFCUCnt + 1
    for i ← 0 to M − 1 do
        CC_Coords[i][0] ← 0
        CC_Coords[i][1] ← 0
    for i ← 0 to N − 1 do
        CC_Coords[FCU_Clust[i]][0] ← CC_Coords[FCU_Clust[i]][0] +
        ((1/clustSize[FCU_Clust[i]]) * rectCenters[i][0])
        CC_Coords[FCU_Clust[i]][1] ← CC_Coords[FCU_Clust[i]][1] +
        ((1/clustSize[FCU_Clust[i]]) * rectCenters[i][1])
    iterCnt ← iterCnt + 1
return CC_Coords, FCU_Clust
```

Algorithm 11: Choose CC places & assign FCUs /w Area Balance.

Data: M, N, innerCorners, rectCenters, rectAreas, X, Y, maxIter
Result: CC_Coords, FCU_Clust

```
movedFCUCnt ← 1
iterCnt ← 0
for i ← 0 to N − 1 do
    oldClust[i] ← 0
    FCU_Clust[i] ← 1
for i ← 0 to M − 1 do
    clustArea[i] ← 0
while (movedFCUCnt > 0) & (iterCnt < maxIter) do
    movedFCUCnt ← 0
    for i ← 0 to N − 1 do
        oldClusters[i] ← FCU_Clust[i]
        minDist ← ∞
        minInd ← 0
        for j ← 0 to M − 1 do
            tempDist ← ((clustersArea[j] + rectAreas[i])/rectAreas[i]) ∗
             sqrt((CC_Coords[j][0] − rectCenters[i][0])^2 +
             (CC_Coords[j][1] − rectCenters[i][1])^2)
            if tempDist < minDist then
                minDist ← tempDist
                minInd ← j
        if iterCnt ≠ 0 then
            clustArea[oldClust[i]] ← clustArea[oldClust[i]] − rectAreas[i]
        FCU_Clust[i] ← minInd
        clustArea[minInd] ← clustArea[minInd] + rectAreas[i]
        if oldClust[i] ≠ FCU_Clust[i] then
            movedFCUCnt ← movedFCUCnt + 1
    for i ← 0 to M − 1 do
        CC_Coords[i][0] ← 0
        CC_Coords[i][1] ← 0
    for i ← 0 to N − 1 do
        CC_Coords[FCU_Clust[i]][0] ← CC_Coords[FCU_Clust[i]][0] +
         ((rectAreas[i]/clustArea[FCU_Clust[i]]) ∗ rectCenters[i][0])
        CC_Coords[FCU_Clust[i]][1] ← CC_Coords[FCU_Clust[i]][1] +
         ((rectAreas[i]/clustArea[FCU_Clust[i]]) ∗ rectCenters[i][1])
    iterCnt ← iterCnt + 1
return CC_Coords, FCU_Clust
```

6.2 Length, Area and Cluster Relations

In Section 6.1, we describe the algorithms used for creating hybrid fog-cloud network topology in order to model our proposed architecture and verify the algorithms to solve IoT data placement problem. In this part, we want to present some experimental results regarding the distributions used while creating FCU and CC network topology.

In Sections 6.2.1 and 6.2.2, distributions of the created rectangles side lengths and the area distributions of the FCUs are given, respectively. In the last Section 6.2.3, we investigate the balance between CC and FCU clusters modeled.

6.2.1 Length Distributions

Details of the point selection algorithm is given in Section 6.1.1, and now we want to investigate the distributions of the lengths generated by selected points. We divide one side of a big rectangle whose side length is '10' into ten smaller pieces, and set the division ratio (perturbation from mean) for each of them is to '0.2' as an example. We repeat the process 10 million times, and we obtain the histogram for the lengths in Figure 6.9. The mean and the variance of the generated lengths are '1.000', '0.016'. On the one hand, mean is somehow expected since we want to obtain ten smaller pieces from a big length of '10'; on the other hand, variance is somewhere between Uniform and Normal distributions. If we consider the first length which is a Uniform distribution and when we keep dividing the rest of big length we end up with a Normal distribution at last for the remaining small piece.

The distribution relation is a corollary of Central Limit Theorem [70]. In each iteration, we generate independent random variables for the smaller lengths, and for the last piece we subtract the sum of these independent random variables from the maximum length which is a constant and has no effect on the distribution. It only affects the mean but not the variance. Mean and variance values for each

length is given in Table 6.1, and the histograms are presented from Figure 6.10 to Figure 6.11.

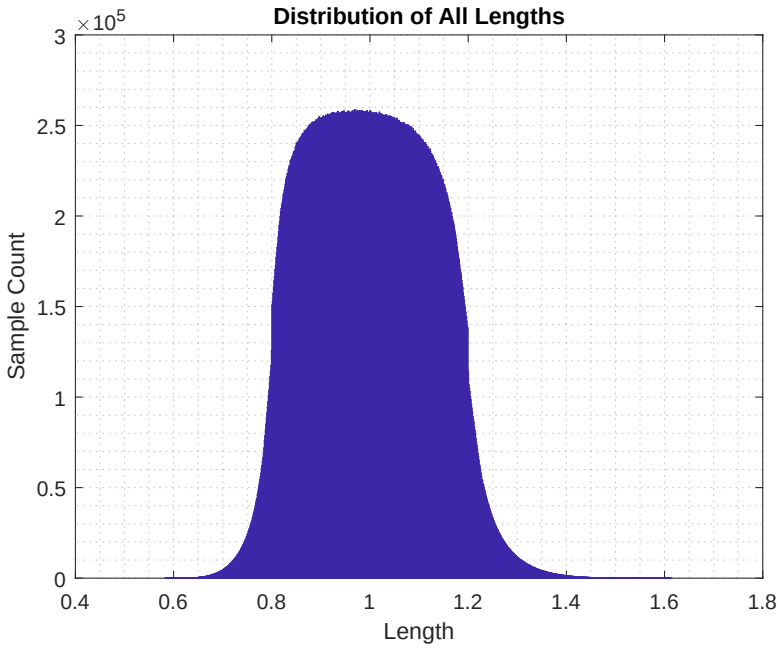
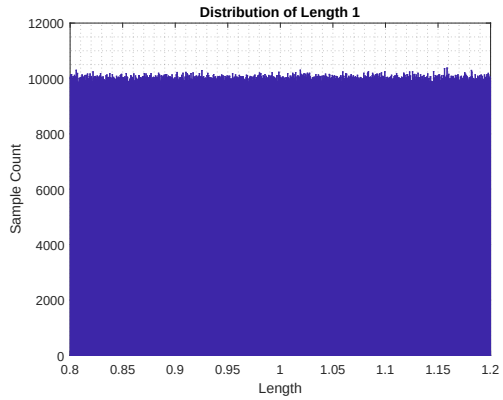


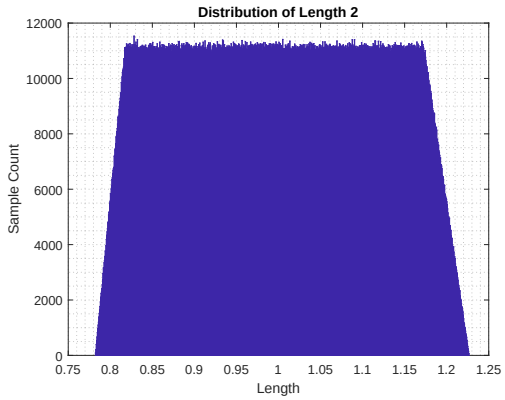
Figure 6.9: Histogram of all lengths in 10 million sample division.

Table 6.1: Mean and variances of generated lengths.

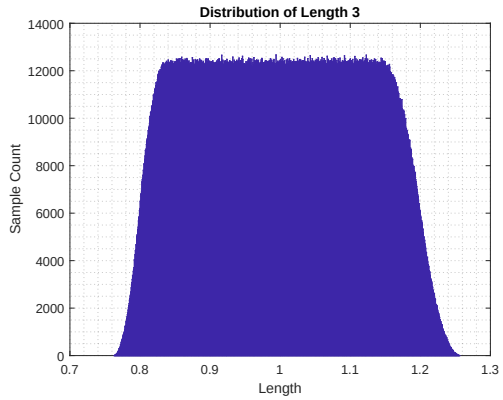
Length #	Mean	Variance
1	1.000	0.013
2	1.000	0.014
3	1.000	0.014
4	1.000	0.014
5	1.000	0.014
6	1.000	0.015
7	1.000	0.016
8	1.000	0.017
9	1.000	0.021
10	1.000	0.021



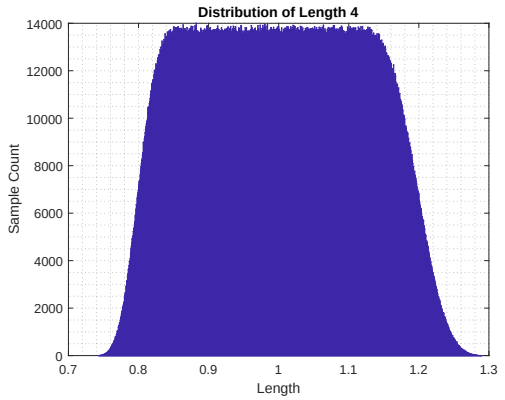
(a) Histogram of Length # 1.



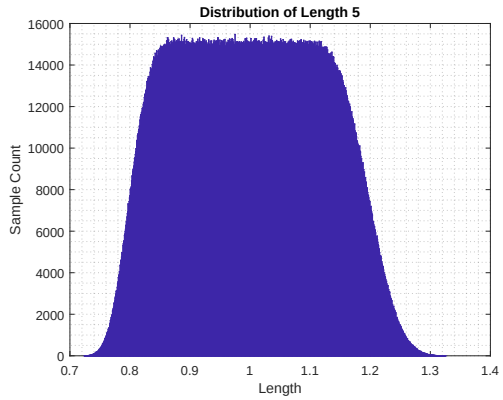
(b) Histogram of Length # 2.



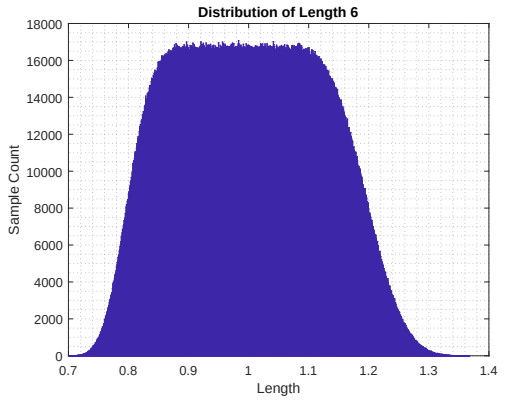
(c) Histogram of Length # 3.



(d) Histogram of Length # 4.

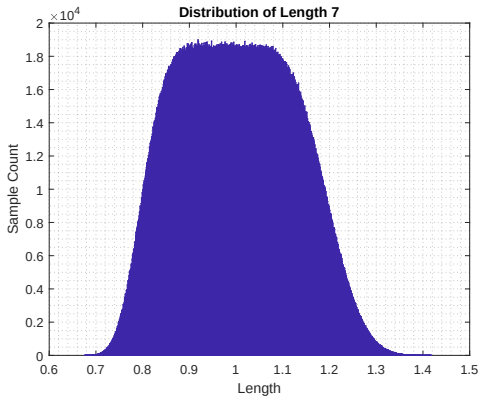


(e) Histogram of Length # 5.

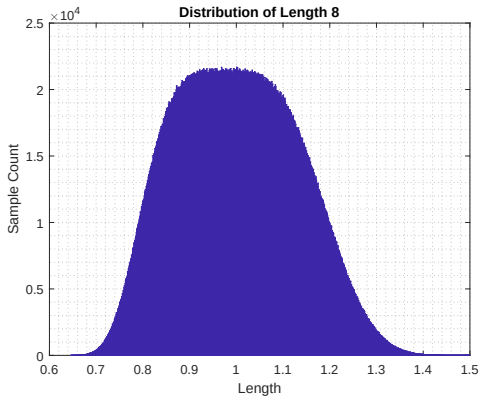


(f) Histogram of Length # 6.

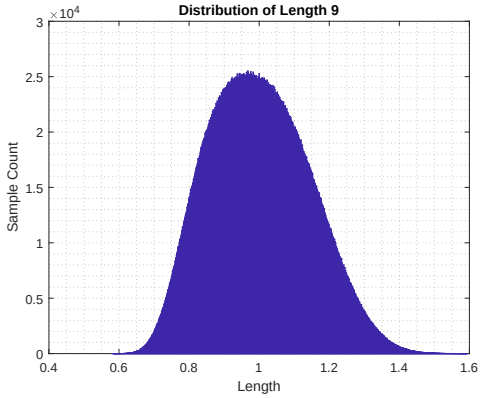
Figure 6.10: Histograms of Length # 1 to # 6.



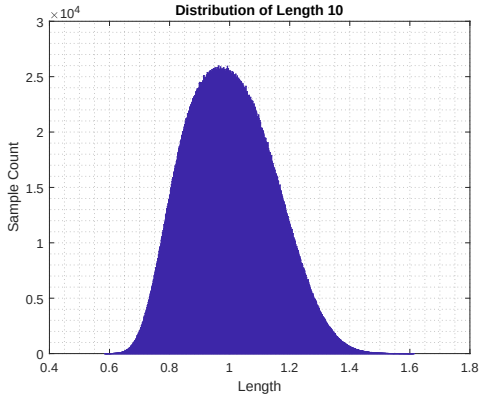
(a) Histogram of Length # 7.



(b) Histogram of Length # 8.



(c) Histogram of Length # 9.



(d) Histogram of Length # 10.

Figure 6.11: Histograms of Length # 7 to # 10.

6.2.2 Area Relationship of Rectangles

In our proposed architecture, we define an architectural element called Fog Computing Unit (FCU) (Chapter 3), and we emphasize that it resembles the neighborhood concept in a city. We also assume that IoT nodes in a FCU are distributed uniformly, so there is a direct correlation between the areas of FCUs and potential data generation and consumption volumes. Hence, in this section we simulate some randomly generated FCUs considering fixed size and FCU count to figure out the relation between the distributions of the generated areas.

We generate a big rectangle whose X & Y lengths are '10-by-5' respectively.

We allow maximum vertical and horizontal cuts of ‘3’ and ‘2’ with a total of ‘6’ FCUs. We give a perturbation ratio for X ‘0.2’ and ‘0.3’ for Y.

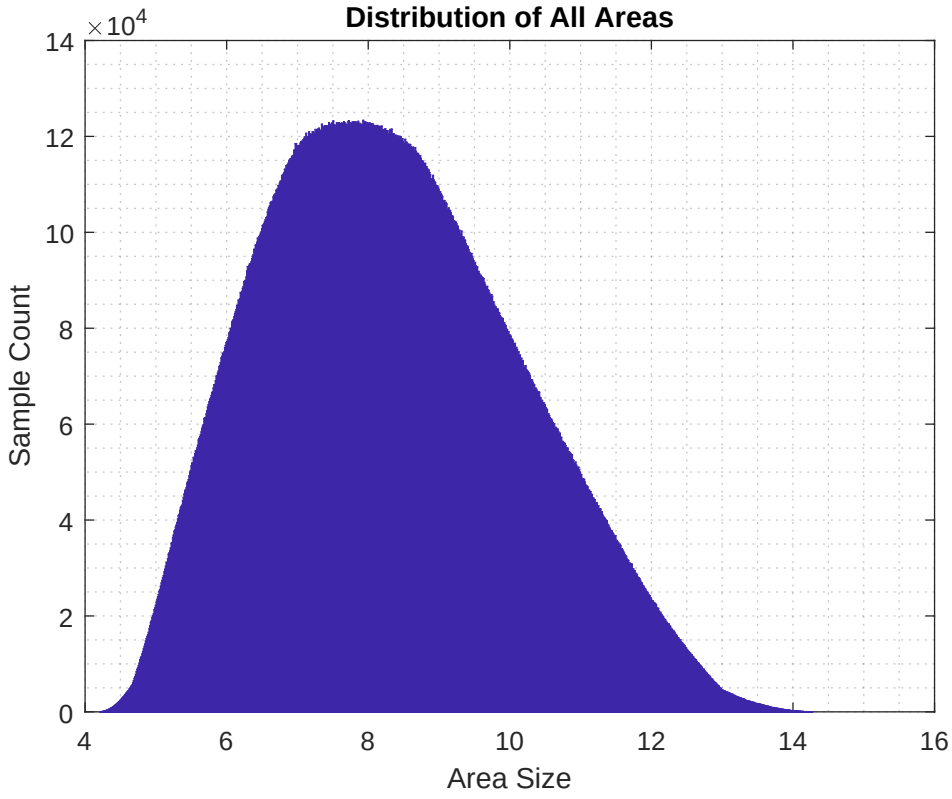
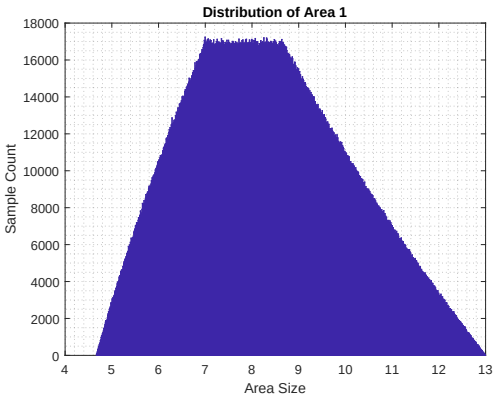


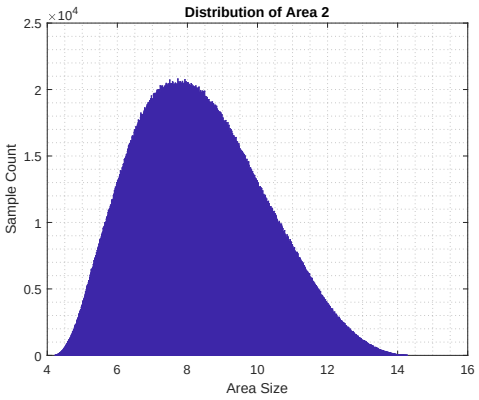
Figure 6.12: Histogram of all areas in 10 million sample division.

Table 6.2: Mean and variances of generated areas.

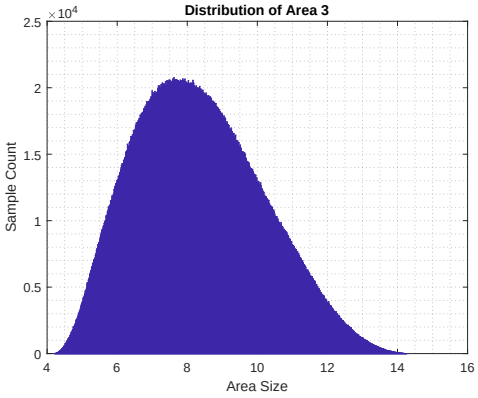
Area #	Mean	Variance
1	8.334	3.037
2	8.334	3.279
3	8.334	3.278
4	8.333	3.036
5	8.332	3.278
6	8.333	3.279



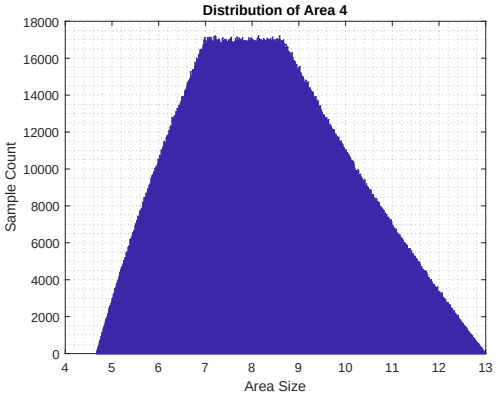
(a) Histogram of Area # 1.



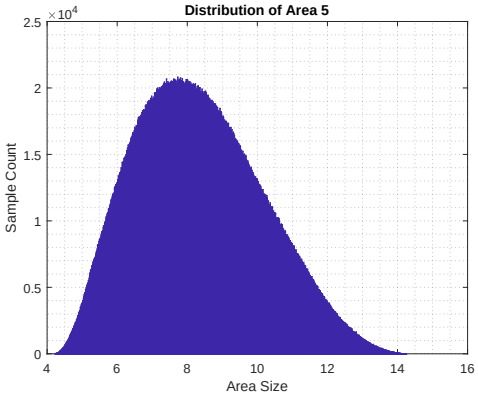
(b) Histogram of Area # 2.



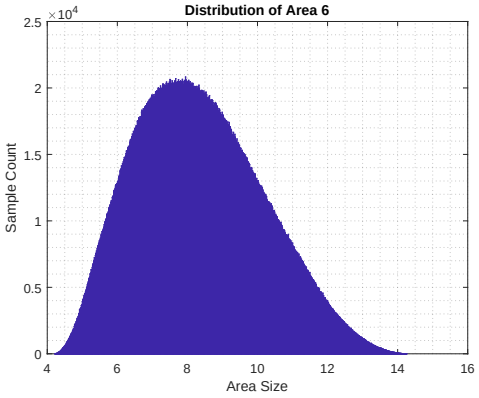
(c) Histogram of Area # 3.



(d) Histogram of Area # 4.



(e) Histogram of Area # 5.



(f) Histogram of Area # 6.

Figure 6.13: Histograms of Areas # 1 to # 6.

6.2.3 Cluster Relationship

In Section 6.1.2, we use k-means for modeling the connections and the assignments between CCs and FCUs with two different strategies: in the first one we connect FCUs to the nearest CC and in the second one we choose less crowded CCs. In this section, we want to give some experimental results regarding to these two different strategies.

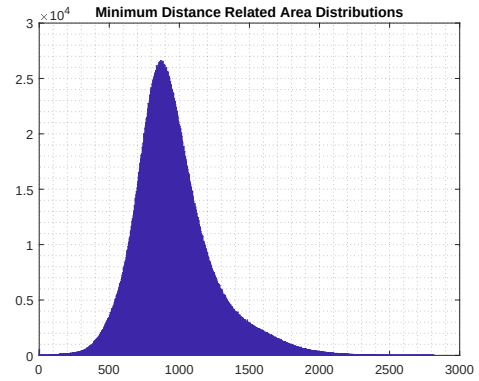
Before giving the results, we want to explain the simulation environment. We model the city as an '80 x 60' rectangle, there exists '40' neighborhoods and we want to place '5' CC data centers. What we expect to see in the simulations is the means of two different strategies should be the same from both area and node count perspectives; but the variances should be different. For the verification, we repeat the process in 1 million different cities with given specs, and we obtain the following results given in Table 6.3 and in Figure 6.14.

Table 6.3: Mean and variances of cluster areas and node counts.

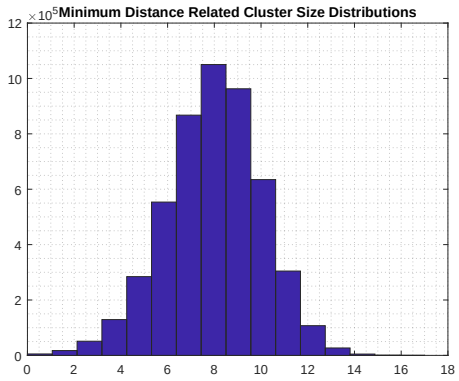
Strategy		Mean	Variance
Minimum Distance	Area	960.000	78024.092
	Node Count	8.000	3.741
Less Crowded CC	Area	960.000	16535.888
	Node Count	8.000	5.493

The simulation results confirm us that the means of both strategies from each perspective are the same: the expected cluster area is equal to the total area divided by the number of cluster count which is '960' according to values chosen, and the mean number of FCUs connected to a CC in each cluster is '8' which equals to the division of the total number FCUs by cluster count.

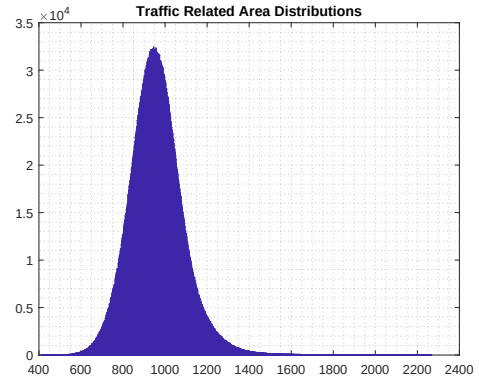
The variances of two strategies from area and cluster node count perspectives are different. It is expected, since when we choose the nearest CC first strategy, we do not consider the cluster area size, but from the node count point of view, the deviation is smaller than the one in less crowded CC strategy. From traffic balancing in a cluster perspective, i.e. less crowded CC strategy, what we expect to see is the areas of the clusters should not deviate much from each other to



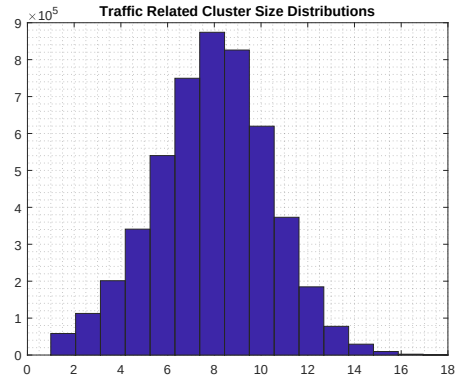
(a) Histogram of Cluster Areas formed by Minimum Distance Strategy.



(b) Histogram of Node Counts formed by Minimum Distance Strategy.



(c) Histogram of Cluster Areas formed by Less Crowded CC Strategy.



(d) Histogram of Node Counts formed by Less Crowded Strategy.

Figure 6.14: Histograms of cluster areas and node counts according k-means strategies.

maintain the balance between clusters. This is what we validate according to the simulation results, the variance of the less crowded strategy is much less that the nearest CC first one. By using this strategy in our models, we sustain the balance between FCU and CC connections from other available services in the network.

6.3 Summary

In this chapter, we try to model the network to be used in the performance evaluations of our proposed solutions since there is no known deployed architecture.

We start the chapter by creating rectangular regions to define the neighborhoods in the smart city, and represent the algorithms related to this (Chapter 6.1.1). In order not to create grid-like neighborhoods, we use a hierarchical cut approach to get smaller rectangles inside the big one.

After the creation of neighborhoods, which we use them as models for FCUs, we try to place cloud computing centers wisely to simulate a possible deployment. To do so, we use a clustering mechanism based on k-means (Chapter 6.1.2) between CCs and FCUs, then decide where to place cloud computing data centers.

In the final part of this chapter, we present some simulation results regarding to the network models created by the algorithms (Chapter 6.2). During the generation of models, we consider the distribution of the neighborhood areas and balance criteria between CCs, since they become very important in a deployed architecture. For the performance evaluation of data placement model and algorithms, we use a model generated by the algorithms presented in this chapter.

Chapter 7

Performance Evaluation

We define the data placement problem in Chapter 3, and present some solution strategies for the problem in Chapters 4 & 5. Now in this chapter, we would like to discuss the performance results of our algorithms and compare them with the optimal solutions. We make extensive simulations using by MATLAB [71] and linear solver Gurobi 7.5.2 [68], which is called inside MATLAB by using its API. We concentrate on four important metrics in our simulations:

- Latency: It is the critical parameter, since we want to place the IoT big data in an effective and efficient way such that the average latency applications encounter minimized.
- Storage cost: It is the second important parameter, while obtaining average minimum latency we do not want to use the data storage elements available in the network excessively.
- Run-time: It is important, since we want to replace linear solvers with heuristic algorithms which give fairly good results when the number of elements in the architecture drastically increase.
- Network occupancy: It is always important, and we do not want to inject more traffic into the network while satisfying the latency demands.

In the rest of this chapter starting with the simulation parameters (Section 7.1) and algorithms used in the evaluation (Section 7.2), we represent the performance of our algorithms according to metrics defined above (from Section 7.3 to Section 7.6).

7.1 Simulation Parameters

For testing the performance of proposed algorithms, we use a network model created by the algorithms given in Chapter 6. We generate a '60 x 80' km^2 smart city consisting of '5' cloud computing data centers (CCs) (**M**). For choosing the places of CCs, we use the second k-means strategy which tries to choose less crowded CCs in order to preserve load balance between CCs. An example of the network model used in the simulations is given in Figure 7.1.

After the generation of the network model, we calculate the delays between FCUs and CCs according to the round trip time of the light [1], assuming that high capacity fiber optic or wireless links exist between data centers. We still need to generate the other parameters explained in Table 4.1. So, we choose different number of applications (**A**) run in the network as '45' and different number of the available data types (**D**) as '20'. For data generation volumes of each data type in each FCU (**GV**), we firstly generate a random variable for the total data generation volume from normal distribution ∼N(1GB, 256MB), then we distribute the generated data volume to each FCU with respect to their areas.

Then we randomly generate applications running frequency in FCUs (**AR**) matrix and their access frequencies to the data types (**AF**) matrix, both, from uniform distribution ∼U[0,3]. For the generation of **AR** matrix, we define a new metric called application run ratio (**appUseRatioInFCU**) which describes at least how many applications run in an FCU, and is is calculated according to the

[1] A request has to be made from one data center to the other and a reply is transmitted back at the speed of light which makes the latency formula as "(2 × Distance) / Speed of light".

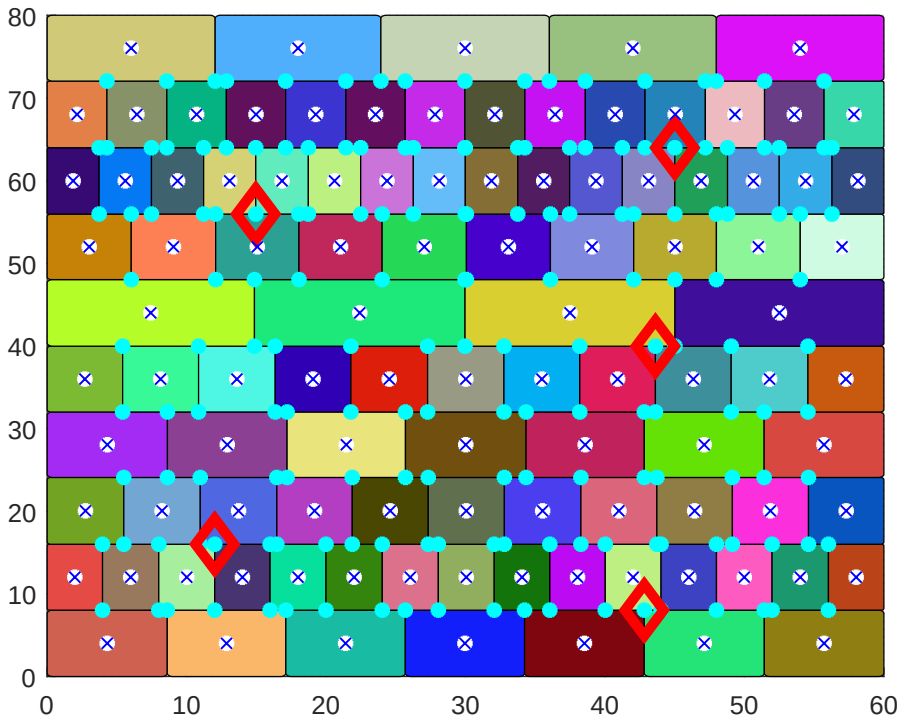


Figure 7.1: Smart city example network: M = 5, N = 100. Blue crosses are FCU centers and cyan points are possible CC locations. After using k-means with less crowded CC strategy chosen CC places are marked with red diamond.

formula given in Equation 7.1.

$$\mathbf{appUseRatioInFCU_i} = \frac{\sum_{j=1}^{A} \mathbf{sgn}(ar_{ij})}{A} \tag{7.1}$$

For a given **appUseRatioInFCU**, **AR** matrix is generated, and it satisfies that at least one application runs in an FCU, and each application runs in at least one FCU. We fix the parameters mentioned above for all simulations except the application run ratio (**appUseRatioInFCU**). For the performance evaluation from different perspectives, we sweep **appUseRatioInFCU** together with other parameters, such as the number of FCUs (**N**), the storage capacities of FCUs (**SC**) and excess use (**EU**) while generating applications data access frequency matrix. For each sweeping variable, we run '100' samples of simulations, and report the averages of these '100' runs as our results.

7.2 Algorithms Used

We compare four proposed algorithms with both random placement, the data-centric and the application-centric solutions in aforementioned smart city network model generated with the parameters given in Section 7.1. For the comparison of the algorithms, we use some abbreviations in the results and these are given in Table 7.1.

Table 7.1: Algorithms used in experiments.

Alg1	The first proposed algorithm
Alg2	The second proposed algorithm
Alg3	The third proposed algorithm
Alg4	The fourth proposed algorithm
RP	Random placement
LPDC	Optimal linear solution for the data-centric placement
LPAC	Optimal linear solution for the application-centric placement

Alg1 is the first placement heuristic algorithm for the data-centric approach (Algorithm 1) which tries to place the mostly generated data type to the nearest data center where it is mostly used. Detailed explanation and pseudo-code are given in Chapter 5.1.

Alg2 is the second placement heuristic algorithm for the data-centric approach (Algorithm 2) which is a slightly different version of the first one as explained in Chapter 5.2.

Alg3 is the third placement heuristic algorithm used in the data-centric approach (Algorithm 3) which considers the cost function as a whole, and tries to minimize it. Detailed explanation of the algorithm is available in Chapter 5.3.

Alg4 is the fourth placement heuristic algorithm (Algorithm 4), which is a modified version of the third one, still considering the defined cost function as whole. Details of the algorithm is given in Chapter 5.4.

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

LPDC denotes the output of linear solver for the model given by the Equations 4.15-4.18 in Chapter 4.1.

LPAC denotes the output of the linear solver for the application-centric approach. It is slightly different version of the **LPDC**. The data is not typed, and the data access frequency denoted by **AF** matrix in the data-centric model is considered in the capacity constraint of the **LPAC** linear model. The corresponding model for **LPAC** is given from Equations 7.2-7.5.

Minimize:

$$\sum_{i \in A} \frac{\sum\limits_{k \in N} ar_{k,i} \sum\limits_{z \in M \cup N} l_{z,(M+k)} p_{i,z}}{\sum\limits_{w \in A} \sum\limits_{q \in N} ar_{q,w}} \tag{7.2}$$

Subject to:

$$p_{i,j} \in \{0, 1\} \tag{7.3}$$

$$\sum_{j \in M \cup N} p_{i,j} = 1, \forall i \in A \tag{7.4}$$

$$\sum_{i \in A} adr_i \times p_{i,(M+j)} \leq sc_j, \forall j \in N \tag{7.5}$$

All the variables used in Equations 7.2-7.5 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 $(\boldsymbol{A} \times 1)$ (Equation 7.6). It is obtained by multiplication of matrix **AF** with vector

DG whose size is $(\boldsymbol{D} \times 1)$ and defined in Equation 4.1.

$$\mathbf{ADR} = \mathbf{AF} \times \mathbf{DG} \tag{7.6}$$

7.3 Latency Results

In this section we give the results related to average latency applications encounter while reaching their required data types defined by Equation 4.4. During the investigation of how average latency is affected, we use two different metrics to be swept together with total number of FCUs exist in the system: 'applications run ratio (**appUseRatioInFCU**)' and 'excess use (**EU**)'. For both metrics we sweep total number of FCUs exist in the network from '50' to '150'. We randomly generate the storage capacities (**SC**) of each FCU from the normal distribution with parameters ∼N(3GB, 1GB), such that an FCU can store '3' different types of data on average.

For making the comparisons of the results easy, we group them in two: comparison of the random placement with first two algorithms as the first group, and as the second group we consider the third and fourth heuristic algorithms together with the outputs of linear solver models explained previously.

7.3.1 Effect of Applications Run Ratio on Latency

In the simulated network model, there exists '45' different applications running requiring at most '20' different data types, which sets minimum excess use (**EU**) to '2.25' (see Equation 4.19), and its maximum value is limited by total number of different applications run in the model. Bearing the values in mind, we set parameter **EU** to a moderate value which is '10' meaning that the available applications require '200' data types in total, not necessarily to be different. We change the application run ratio in an FCU from '0.05' to '1.0'.

Comparisons of the algorithms are given in the next two sections (Section 7.3.1.1 & 7.3.1.2).

7.3.1.1 Comparison of Alg1 & Alg2 & RP

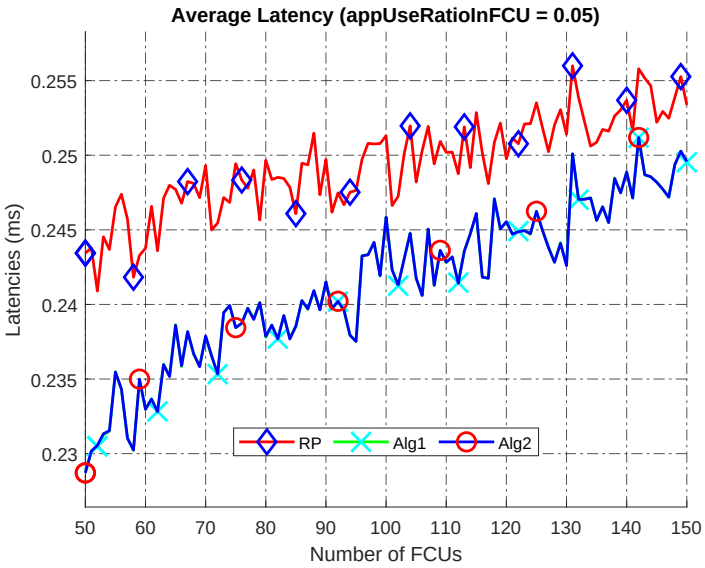


Figure 7.2: Average latency (EU = 10, appUseRatioInFCU = 0.05).

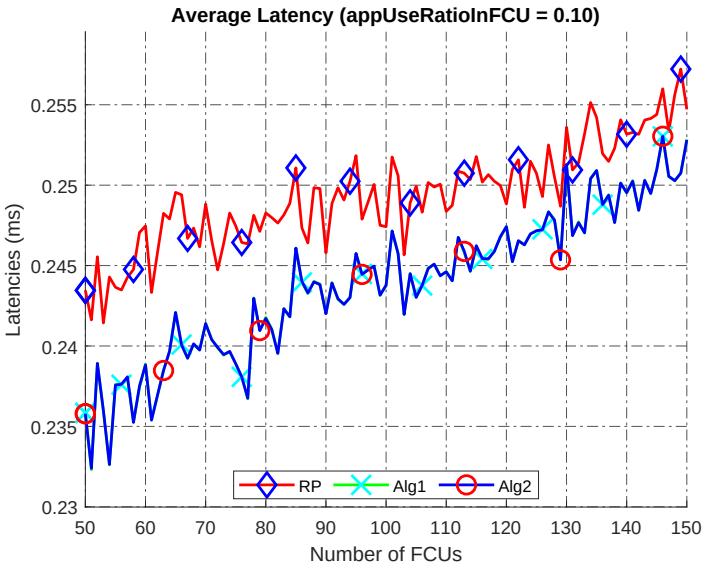


Figure 7.3: Average latency (EU = 10, appUseRatioInFCU = 0.1).

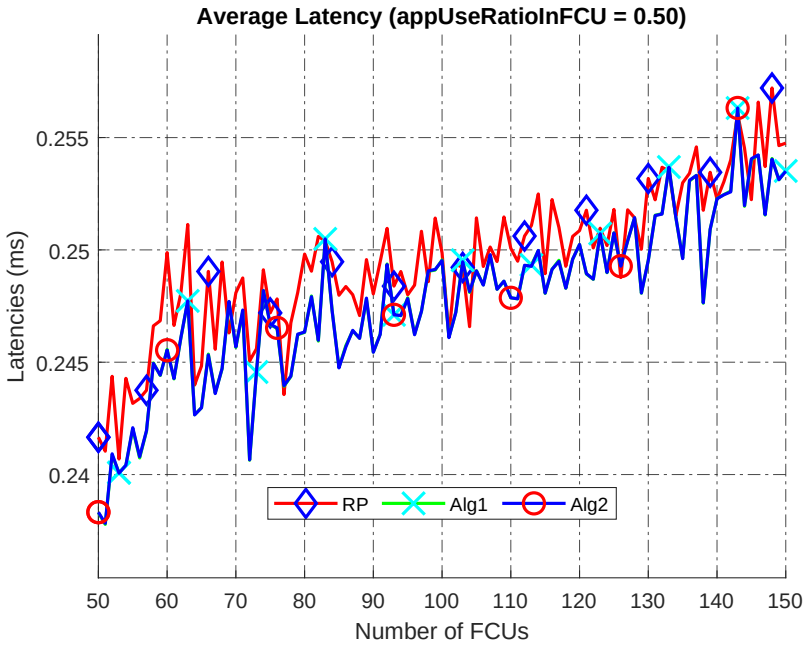


Figure 7.4: Average latency (EU = 10, appUseRatioInFCU = 0.5).

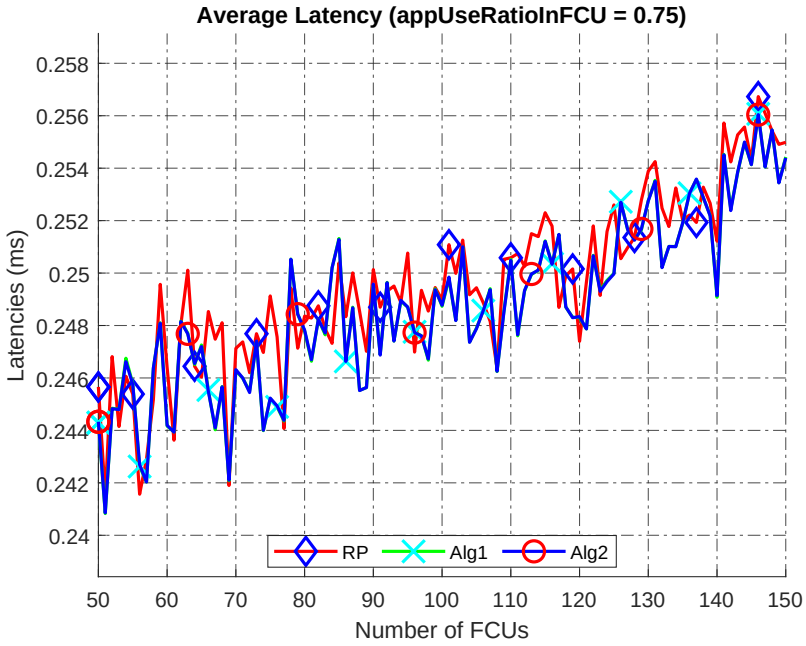


Figure 7.5: Average latency (EU = 10, appUseRatioInFCU = 0.75).

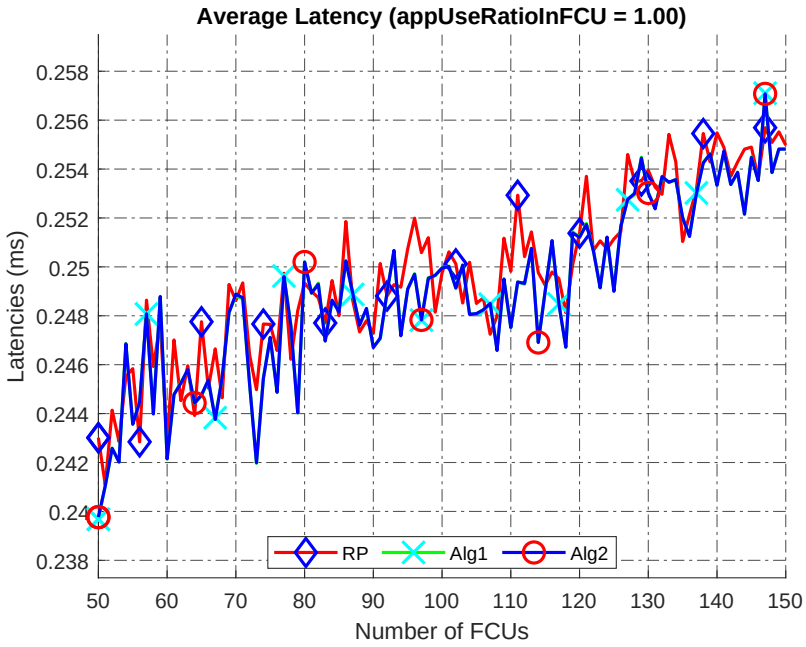


Figure 7.6: Average latency (EU = 10, appUseRatioInFCU = 1.0).

In this group of algorithms, we do not consider the average latency cost function as a whole, and all the results are worse than the algorithms in the second group as seen from Figures 7.2 to 7.11.

The average latency encountered by applications increases as the number of FCUs grows. There are two reasons of this, at first when the number of FCUs scales up, average distances between data centers increase, and as the second reason the probability of placing data to a far location where it is used moderately increases.

If we compare the algorithms with each other our first two algorithms give better results when the application run ratio (**appUseRatioInFCU**) is less than '0.5' (see Figures 7.2- 7.4). After '0.5' all the algorithms give almost the same result. This is somehow expected, since when **appUseRatioInFCU** approaches to '1', the number of applications running on an FCU increases and it becomes problematic to choose the place for a data type where it is mostly used. Although our first two proposed algorithms are far from the second group, they give better or similar results as the random placement.

7.3.1.2 Comparison of Alg3 & Alg4 & LDC & LPAC

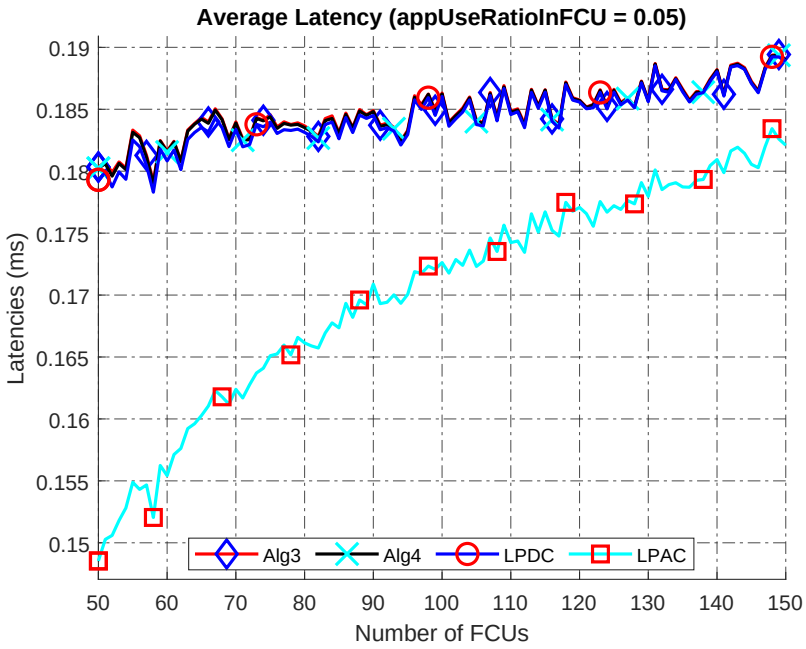


Figure 7.7: Average latency (EU = 10, appUseRatioInFCU = 0.05).

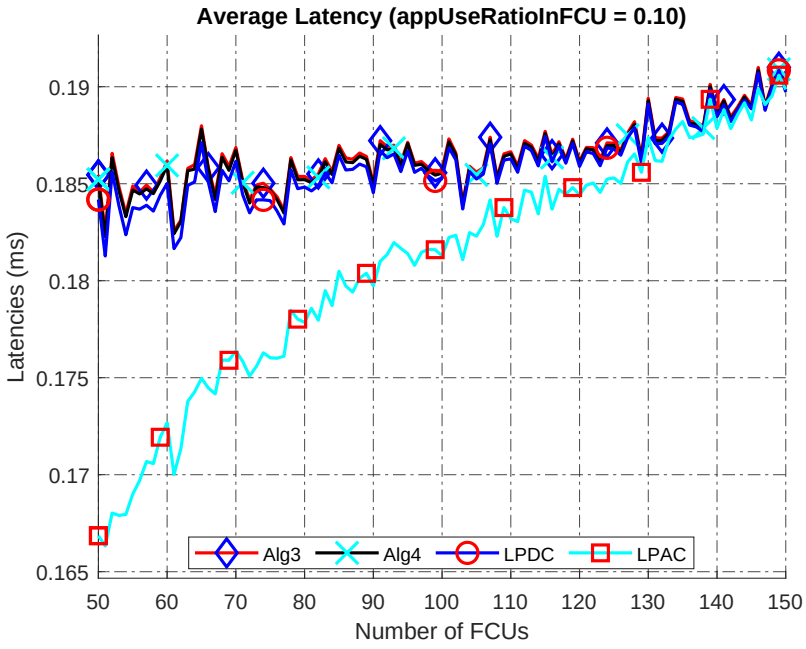


Figure 7.8: Average latency (EU = 10, appUseRatioInFCU = 0.1).

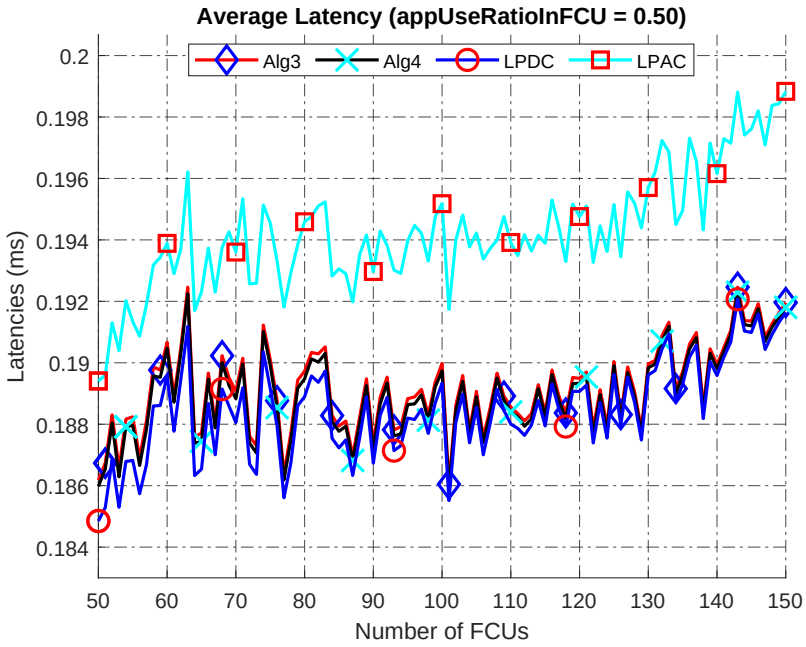


Figure 7.9: Average latency (EU = 10, appUseRatioInFCU = 0.5).

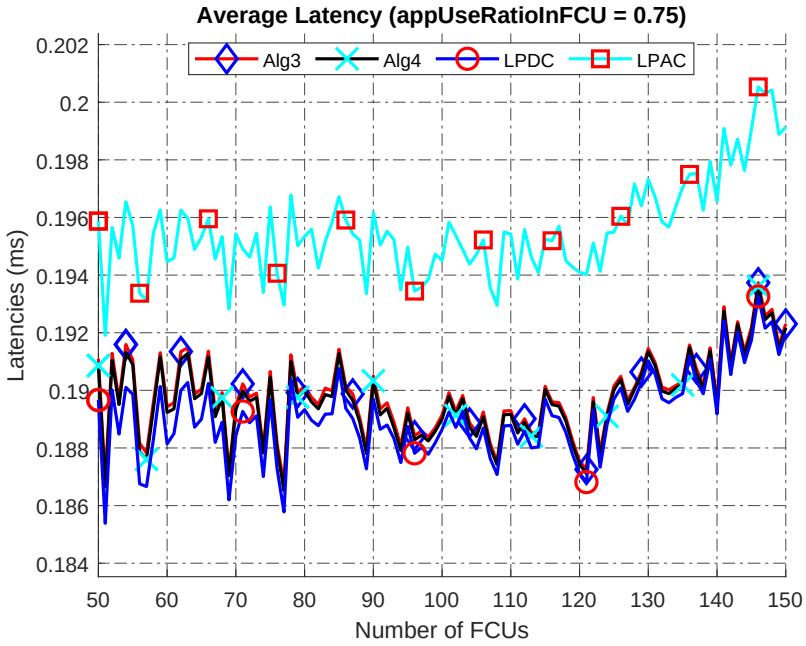


Figure 7.10: Average latency (EU = 10, appUseRatioInFCU = 0.75).

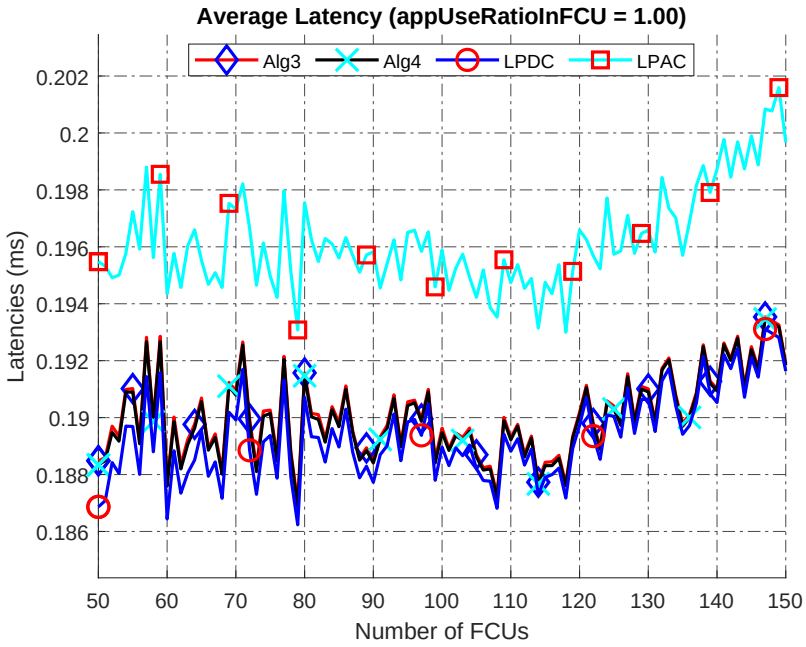


Figure 7.11: Average latency (EU = 10, appUseRatioInFCU = 1.0).

These algorithms consider the defined average latency cost function as a whole, and as a consequence of that they give almost '30%' better results than the random placement and our proposed first two algorithms. We also evaluate the linear model of the application-centric (**LPAC**) data placement with this group, and its results are better when **appUseRatioInFCU** and the number of FCUs are small. Although it gives good results with small number of applications running in FCUs and less crowded architectures, its performance changes drastically as these two numbers grow. As seen in Figure 7.11, its performance is '4%' less than the data-centric approaches.

If we compare the average latencies of the data-centric approaches, the best result is obtained by **LPDC** which is inevitable, since it is the optimal solution of the defined model and solved by a linear solver, Gurobi [68]. When the number of FCUs increase, the average latency of all data-centric approaches also increase, but the rise is not as high as in the case of the application-centric one. As the simulation results confirm, our proposed third and fourth algorithms give almost the same results as the optimal solution, and their performances for this setup are almost the same. As seen in the Figures 7.7- 7.11, they track the optimal

solution with a slight difference.

Before finishing the discussion about the effect of application run ratio on latency, we want to consider a final scenario in which FCU count increases drastically, and all applications run in each of the FCUs. It is a possible scenario when Social Internet of Things (SIoT) [72] concept is deployed, and becomes widely used. So instead of sweeping the FCU count '50' to '150', we continue rising it until '500' and set application run ratio to '1.0', meaning all '45' of '45' applications run in all FCUs (see Figure 7.12& 7.13).

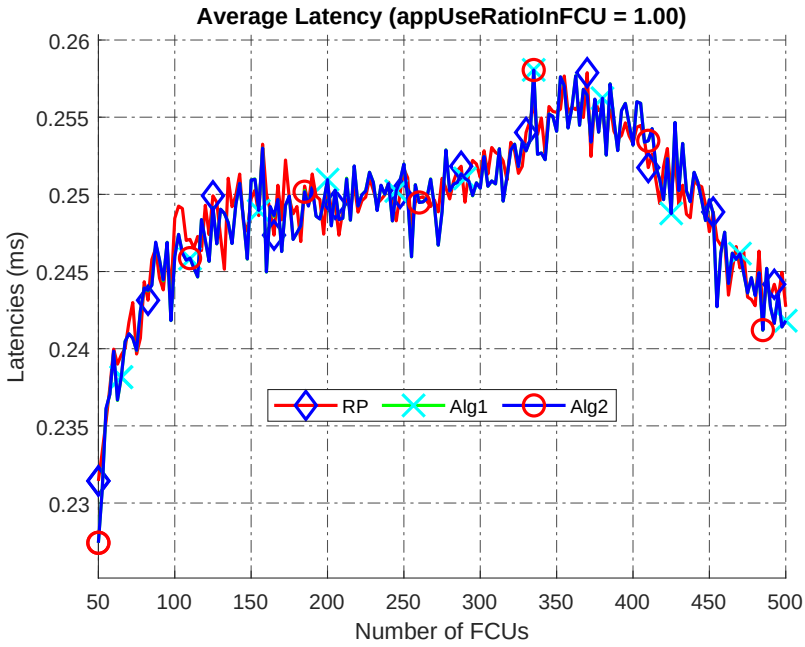


Figure 7.12: Average latency (EU = 10, appUseRatioInFCU = 1.0).

The interesting thing in this scenario is at some point average latencies start to decrease. This can be expected as there is a limited rectangular area, which is '60 x 80' km^2, and it is divided into many more small rectangles such as 500 smaller ones. Therefore, the average distance between the centers of these rectangles start to decrease, so the average latency does. Also again in this case, the data-centric approach performs better than application-centric placement by about '10%'.

All simulations verify that for networks composed of FCUs and CCs, if running frequencies of applications on FCUs are high, the data-centric placement approach

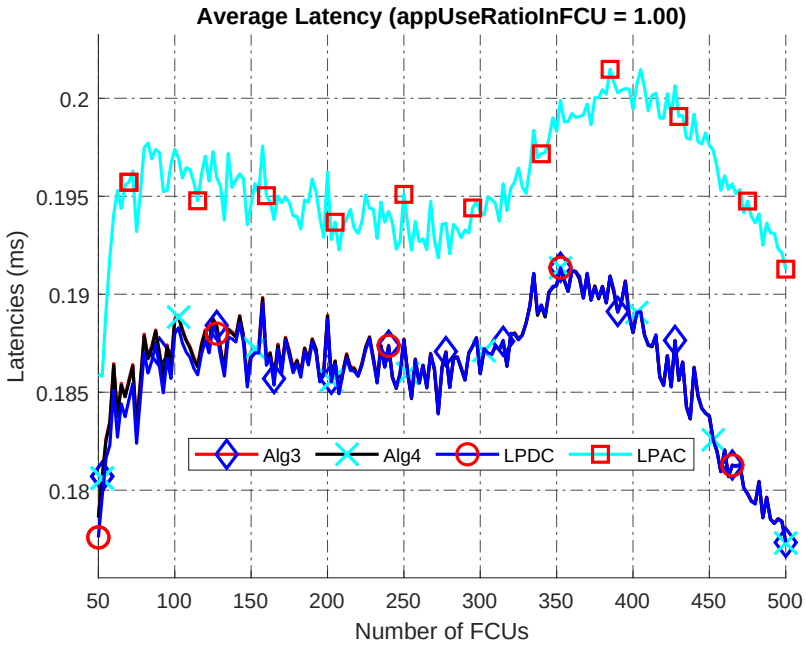


Figure 7.13: Average latency (EU = 10, appUseRatioInFCU = 1.0).

outperforms the application-centric one, and our proposed algorithms **Alg3** and **Alg4** are good alternatives to be used instead of the linear model.

7.3.2 Effect of Excess Use on Latency

From average latency perspective, we investigate the effect of excess use metric in this section. It is an indication of the dependencies between the available data types and applications. There is a linear proportion with this metric and the required number of data types by the applications (see Section 4.2).

We fix the number of different applications running to '45', and for these simulations we choose **appUseRatioInFCU** to '0.2' which means that '9' different applications run in an FCU. Also the number of available different data types in the model is set to '20', and this limits the minimum excess use to '2.25'. By considering these, we sweep excess use parameter from '2.5' to '10' while changing the total number of FCUs from '50' to '150'.

7.3.2.1 Comparison of Alg1 & Alg2 & RP

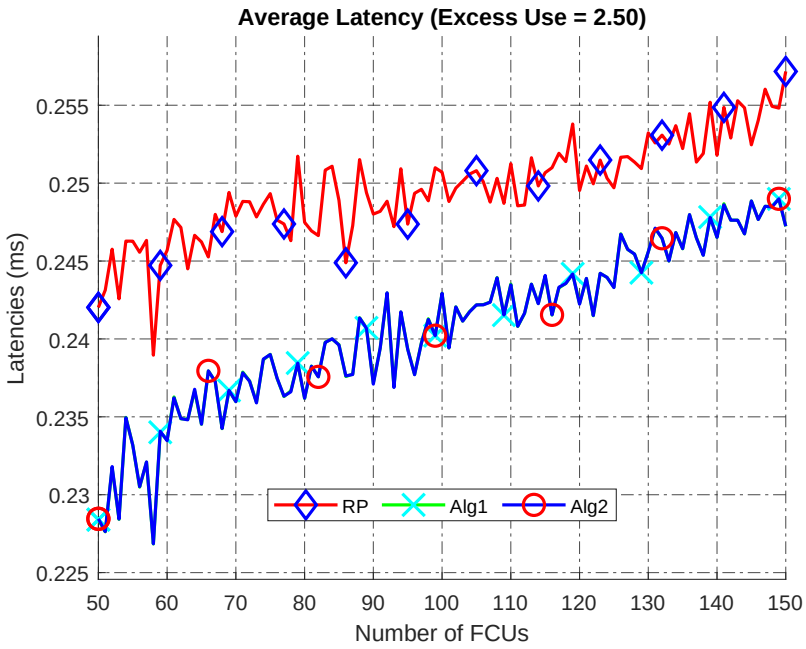


Figure 7.14: Average latency (appUseRatioInFCU = 0.2, EU = 2.5).

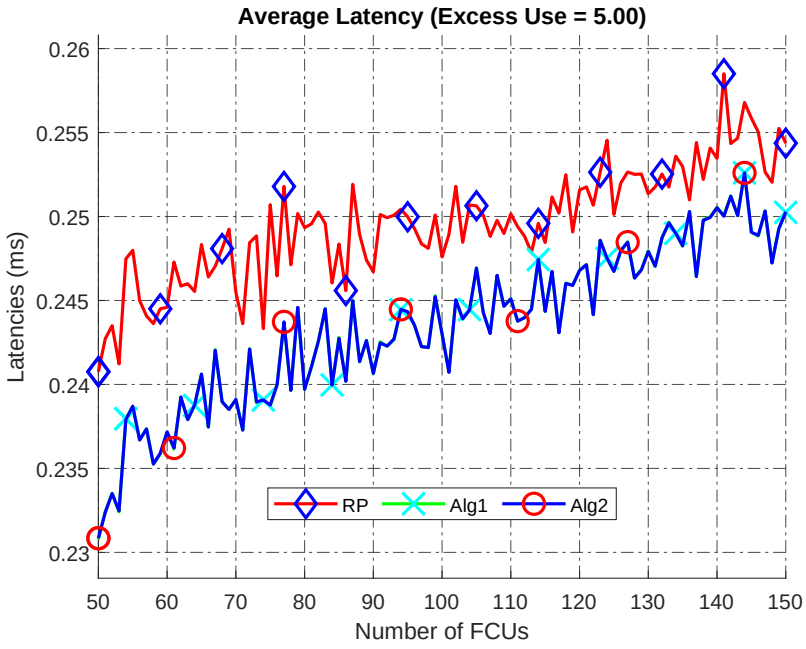


Figure 7.15: Average latency (appUseRatioInFCU = 0.2, EU = 5.0).

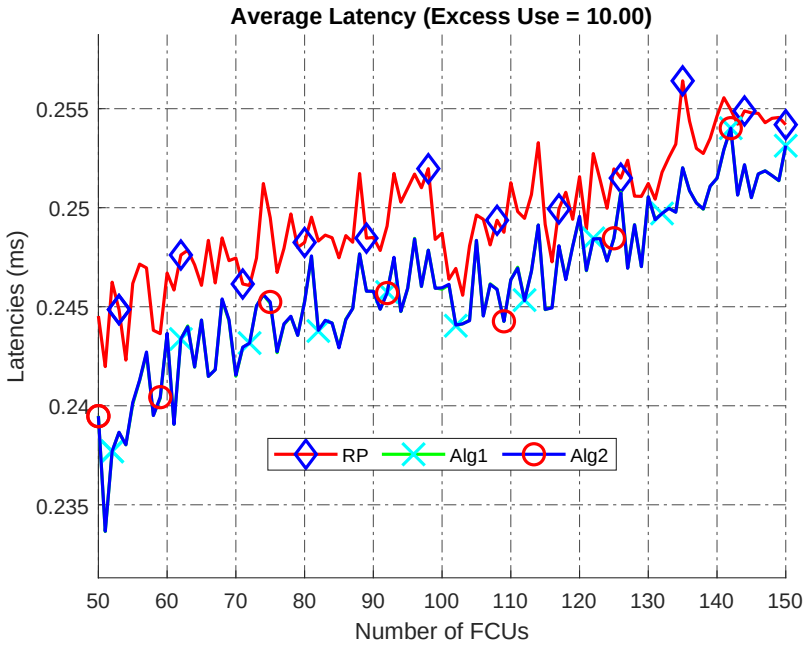


Figure 7.16: Average latency (appUseRatioInFCU = 0.2, EU = 10.0).

In the first group of the simulations, we compare the random placement with our first two algorithms (Figures 7.14- 7.16). The results resemble what we observe in **appUseRatioInFCU** simulations, as excess use increases the performance of two heuristics degrades, but they are still better than the random placement. It is expected since we increase the dependency on each data type by increasing the excess use parameter. This is the difference between **appUseRatioInFCU** case, where we increase the number of applications in an FCU.

Both algorithms in this group are also worse than the second group, because they just try to minimize one of the maximum delays encountered by an application for reaching its required data.

7.3.2.2 Comparison of Alg3 & Alg4 & LDC & LPAC

In this group we discuss the effect of excess use on our remaining algorithms together with the linear solver results (Figures 7.17- 7.19).

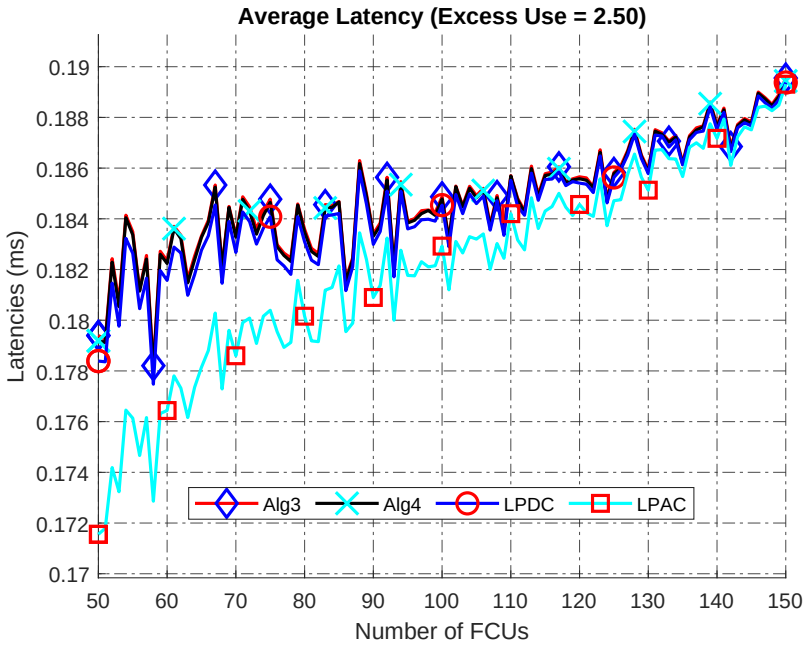


Figure 7.17: Average latency (appUseRatioInFCU = 0.2, EU = 2.5).

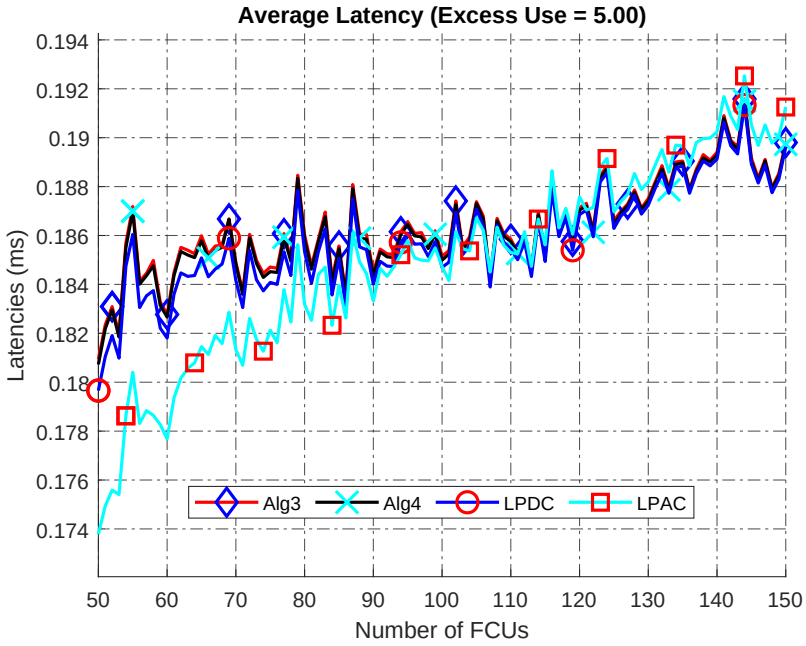


Figure 7.18: Average latency (appUseRatioInFCU = 0.2, EU = 5.0).

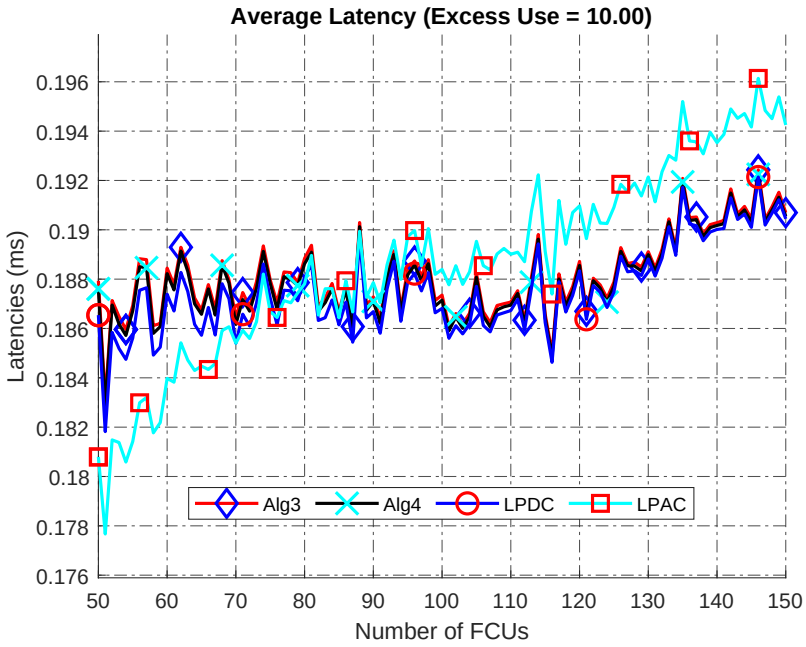


Figure 7.19: Average latency (appUseRatioInFCU = 0.2, EU = 10.0).

At first, we want to discuss the application-centric data placement (**LPAC**). According to the results obtained, as our sweeping parameter increases the performance of the application-centric placement degrades. It is somehow expected from storage perspective, and this is the problem we want to solve with the data-centric approach. As excess use increases, the dependencies of the data types also rises which create pressure on the storage costs. When dependency on data types for an application increases, required storage capacity also expands. Regarding this fact, it becomes difficult to place an application's whole data in an FCU, and these data types need to choose CC data centers to be placed.

Performance of the data-centric approaches slightly degrades as the parameter increases. Also when the number of FCUs expands, there is a rise in the average latency of both heuristic algorithms and the data-centric linear solution. The reason is the same as in the case of **appUseRatioInFCU** simulations. When the number of the FCUs increases, the total number of applications running in the network also increases, and this makes difficult to find the best place of all running applications distributed among FCUs.

Now, we want to compare our heuristics with the output of linear solver. It is clear that all possible solutions are bounded by the output of linear solver, since it is optimal. What we try to do is to obtain a solution which is as close as to the optimal one, and definitely it is the case. Algorithms **Alg3** and **Alg4** both track the optimal solution and they give almost '0.1%' worse results on the average.

To sum up, all the latency related discussions with considering all the simulations, the data-centric approaches overwhelm the application-centric one, when there exists highly data dependent applications running. And our proposed algorithms **Alg3** and **Alg4** are fair replacements of linear model. If we consider our proposed **Alg3** and **Alg4** algorithms in all cases, **Alg4** is slightly better than the **Alg3** in average latency perspective, but it has a drawback compared to **Alg3** which becomes clear when we discuss the run-times of the algorithms.

7.4 Storage Results

In this section, we discuss where data types are placed when using the algorithms. We divide the possible storage places into two: CCs and FCUs. During the simulations how the data placement is affected, we consider three different metrics swept together with the number of available FCUs. As a first metric we consider again **appUseRatioInFCU** which is responsible for how many applications run in an FCU (Section 7.4.1). Excess use is considered as the second parameter as an indication of the dependency between data types and running applications (Section 7.4.2). Finally, we consider the available storage capacities (**SC**) of the FCUs (Section 7.4.3).

For better presentation of the results in this group, we divide the figures into two. In the first figure, we present results of **LPAC**, **LPDC** and random placement (**RP**). In the second figure, we give results all data-centric approaches except **RP**.

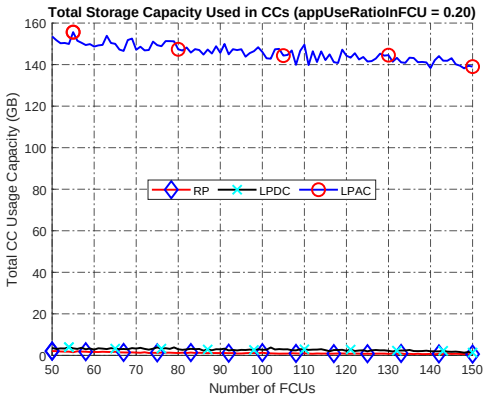
7.4.1 Effect of Applications Run Ratio on Data Storage

We start with the effect of application running ratio on the required storage capacities. Since we also consider this parameter in the average latency (Section 7.3.1), in this section we want to see whether a correlation between used storage capacities and average latency exists. Previously, we conclude that if the number of FCUs increases, the average latency also increases in all cases. When we choose a higher **appUseRatioInFCU** parameter, the latency increases but the performance of the application-centric strategy becomes worse than the data-centric ones.

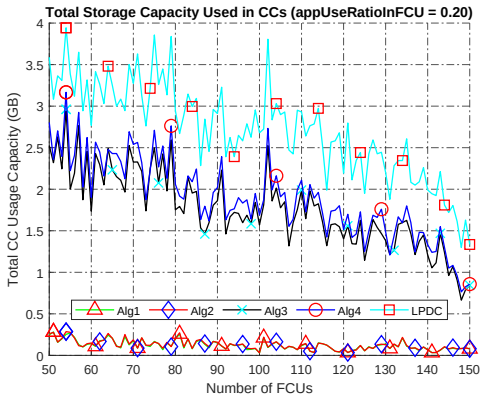
In this section, we want to see whether the effect of the increase in the latency in the application-centric placement is a consequence of where the data is placed or not. We also want to observe how the proposed data-centric approaches place data according to the parameter **appUseRatioInFCU**. We fix **EU** to '5.0', and the storage capacity of an FCU is randomly chosen such that it can store 3 different types on the average from the normal distribution $\sim$N(3GB, 1GB).

There are two dimensions in the figures (Figure 7.20- 7.22): the number of the FCUs and **appUseRatioInFCU** parameter. It is clear since the total required capacity is constant, there is an inverse proportion between the used storage capacities in CCs and FCUs. In the discussions we consider used storage capacities in the FCUs, the converse of all the comments is true for the CCs.

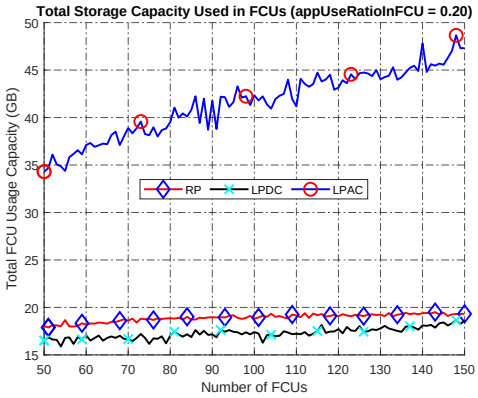
Firstly, we focus on the number of FCUs. In all cases simulated, as it can be seen from the figures that there is a direct proportion between the FCU count and used storage capacity in the FCUs. This is a consequence of increasing the possible places for data to be placed. Secondly, we consider **appUseRatioInFCU** parameter, and the result we obtain is the contrary of previous case. When we increase this parameter, used storage capacity in FCUs decreases. We can explain this as the correlation between latency and the number of applications running in FCUs. When a data type is requested from many places, it has to be placed such that it can be reached from all places requested easily, and this can be achieved by putting it on somewhere in the middle.



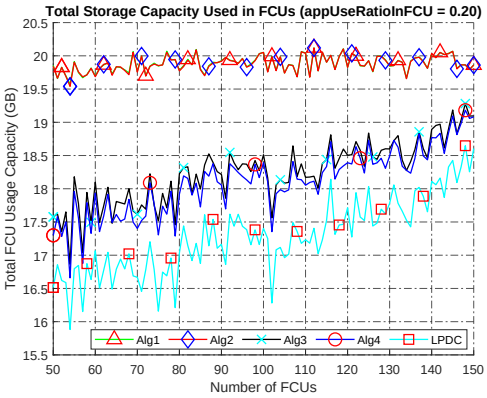
(a) Used CC Capacity in LPAC, LPDC and RP.



(b) Used CC Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.



(c) Used FCU Capacity in LPAC, LPDC and RP.

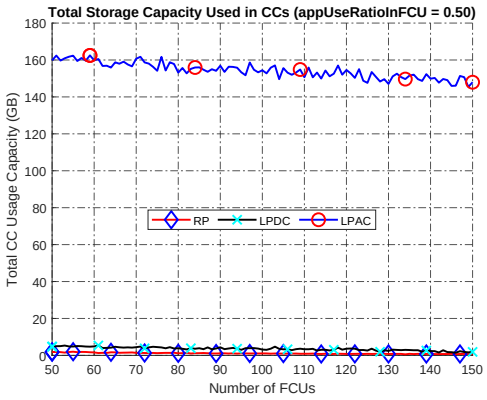


(d) Used FCU Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.

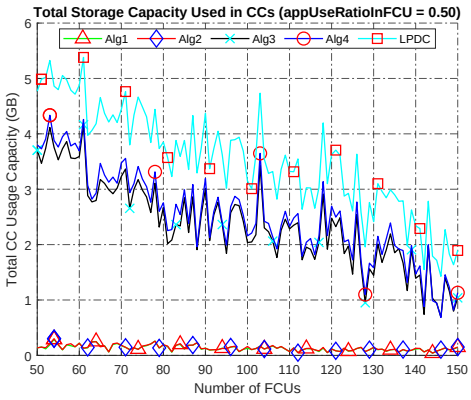
Figure 7.20: Total used CC & FCU storage capacities (appUseRatioInFCU = 0.2, EU = 5.0).

When we consider the algorithms used in the simulations individually, what we see is **LPAC** requires the most data storage total as expected. It is higher than '180GB' and this is inevitable, since all applications replicate their required data type instead of sharing. On the contrary, the data-centric placement strategies only require '20GB' which is directly related with total number of different data types exists in the network model. Hence, the data-centric approaches give much more better results than traditional placement approach.

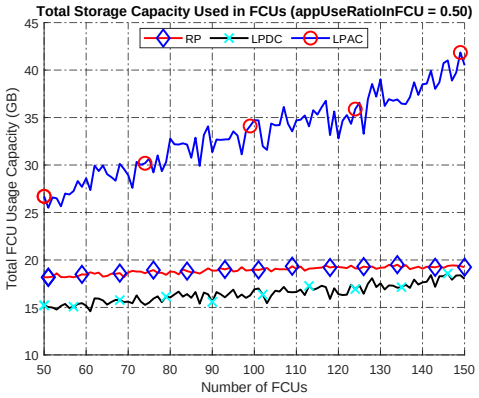
First two data-centric heuristic algorithms choose FCUs frequently than third and fourth ones, since they do not consider the whole latency. In other approaches



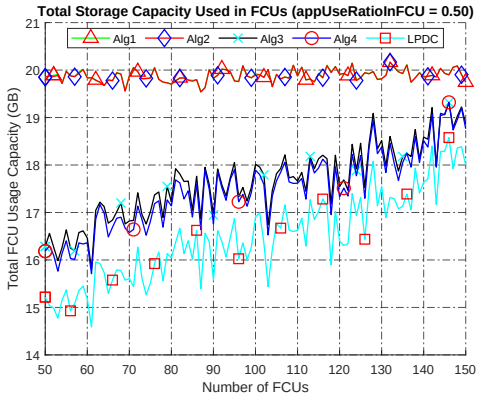
(a) Used CC Capacity in LPAC, LPDC and RP.



(b) Used CC Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.



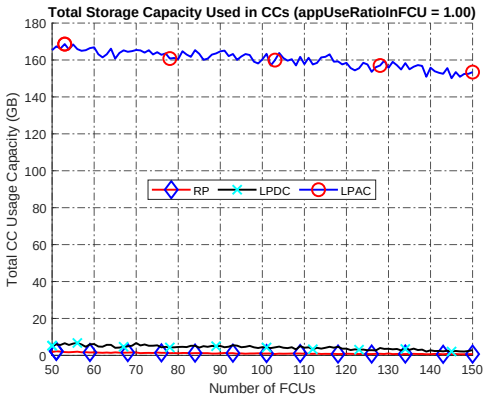
(c) Used FCU Capacity in LPAC, LPDC and RP.



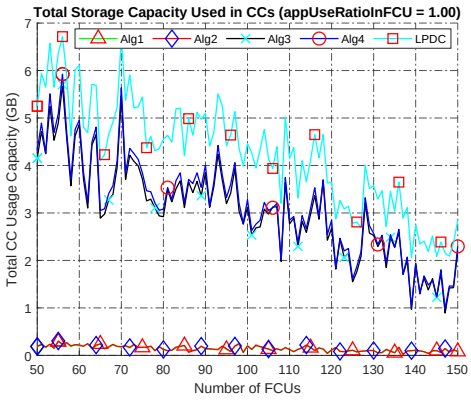
(d) Used FCU Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.

Figure 7.21: Total used CC & FCU storage capacities (appUseRatioInFCU = 0.5, EU = 5.0).

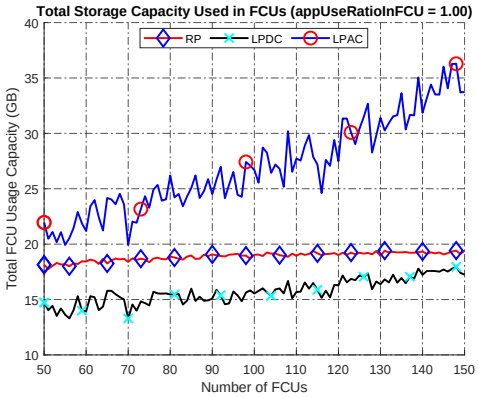
except these two, there exists a balance between data to be stored in FCUs and CCs. The ratio of used capacities of the FCUs increases '20%', when the total number of FCUs increases. This verifies that in a crowded network model, the data-centric approaches try to store the data in the FCUs. It also supports the result obtained in the average latency case: when the parameter increases, it becomes harder to find the best place among the FCUs, and as a consequence the average latency increases.



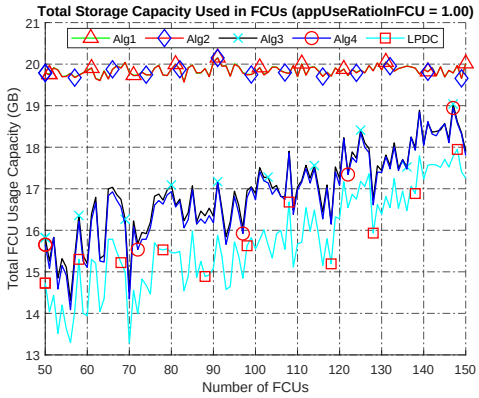
(a) Used CC Capacity in LPAC, LPDC and RP.



(b) Used CC Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.



(c) Used FCU Capacity in LPAC, LPDC and RP.

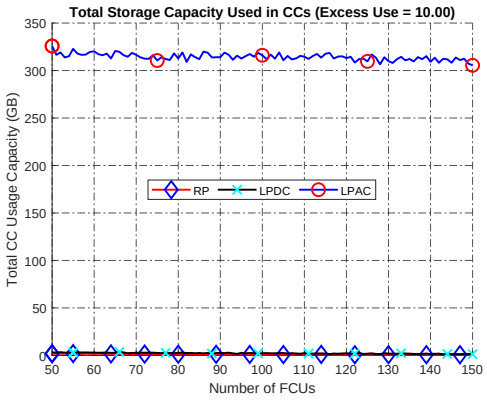


(d) Used FCU Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.

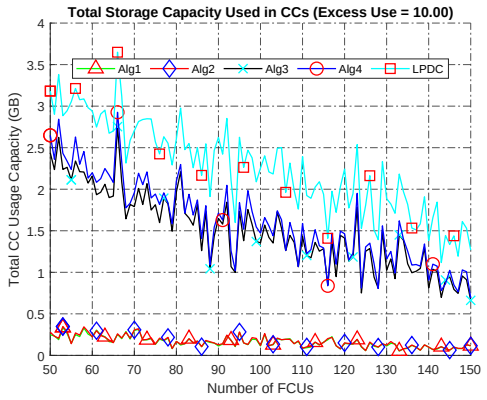
Figure 7.22: Total used CC & FCU storage capacities (appUseRatioInFCU = 1.0, EU = 5.0).

7.4.2 Effect of Excess Use on Data Storage

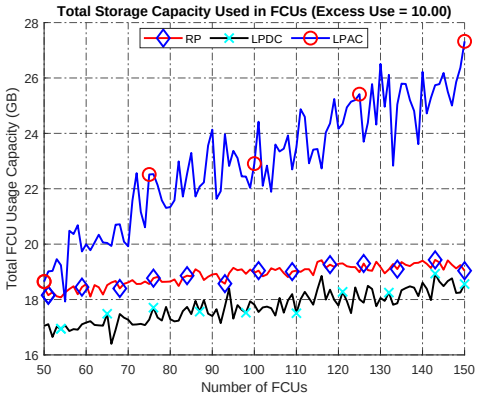
Now, we want to discuss what happens if the dependency between the data types and applications increases, and the parameter that represents this is excess use (**EU**). In Section 7.3.2, we explain how the latency is affected regarding to this parameter, and now we want to figure out whether there exists a relation between the required storage capacities and the latency. From the latency perspective, we conclude there is a direct proportion between excess use and latency: when excess use increases, the latency also increases; but it is more sensible in while



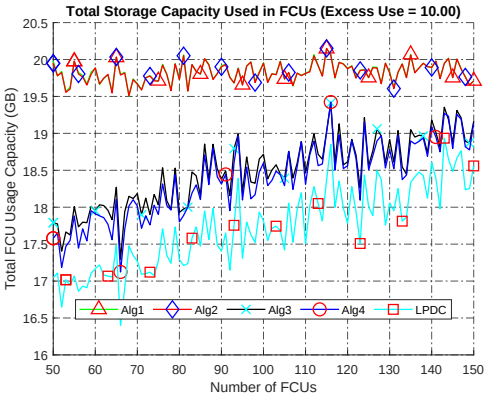
(a) Used CC Capacity in LPAC, LPDC and RP.



(b) Used CC Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.



(c) Used FCU Capacity in LPAC, LPDC and RP.



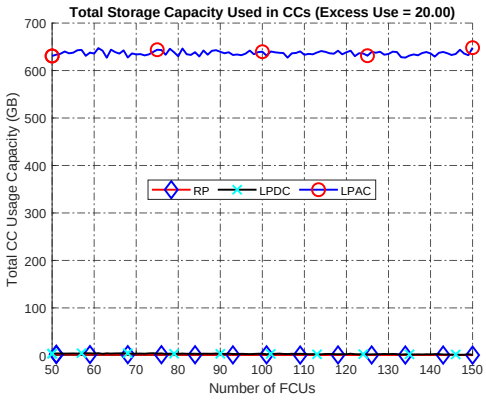
(d) Used FCU Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.

Figure 7.23: Total used CC & FCU storage capacities (appUseRatioInFCU = 0.05, EU = 10.0).

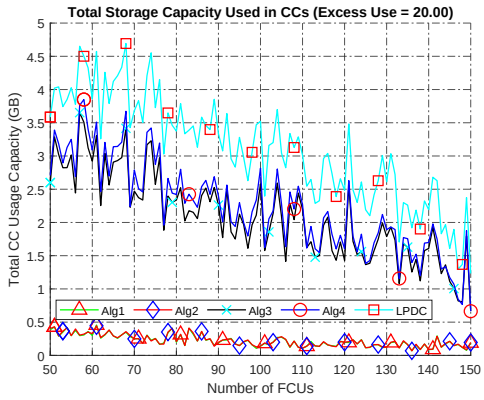
using **LPAC** strategy than the data-centric approaches.

As we have shown in previous section that there is an inverse proportion between **appUseRatioInFCU** and the storage capacity used in FCUs. So for this purpose we choose a smaller value of the parameter we use in the previous section (**appUseRatioInFCU**) and set it to '0.05' for these simulations. Storage capacity of an FCU is again chosen randomly from ∼N(3GB, 1GB) distribution.

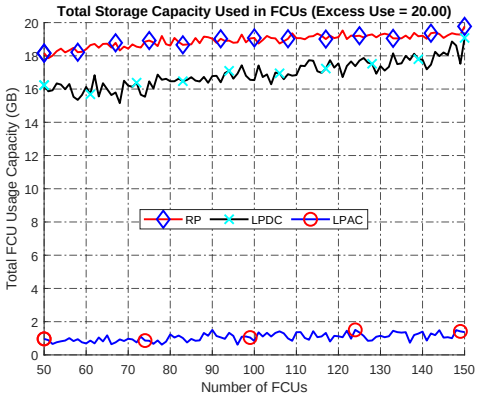
We use two different parameters to be swept in the simulations: the number of the FCUs and excess use. We sweep FCU count from '50' to '150' in all



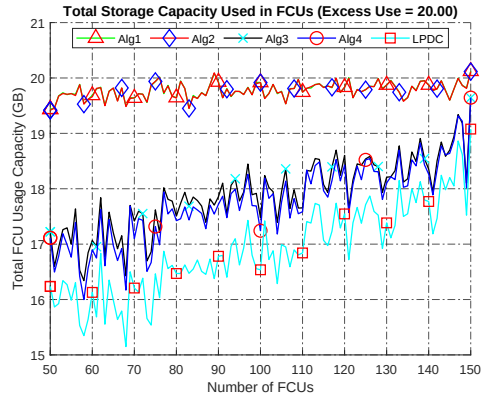
(a) Used CC Capacity in LPAC, LPDC and RP.



(b) Used CC Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.



(c) Used FCU Capacity in LPAC, LPDC and RP.

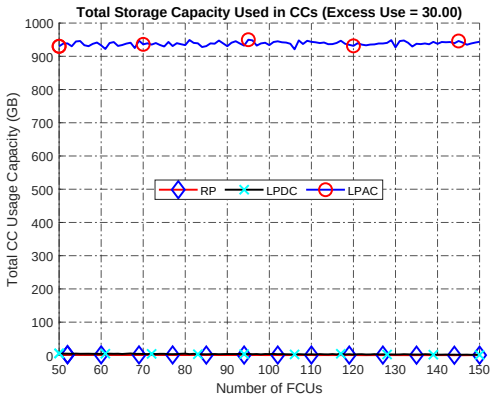


(d) Used FCU Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.

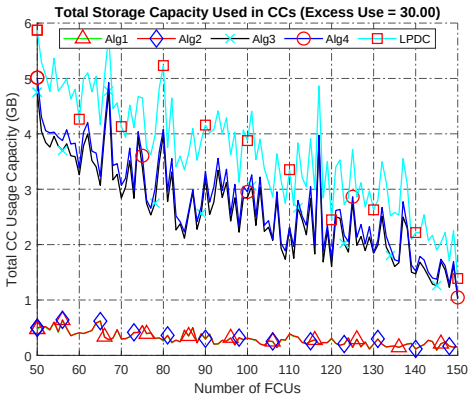
Figure 7.24: Total used CC & FCU storage capacities (appUseRatioInFCU = 0.05, EU = 20.0).

simulations, and for excess use we use '10', '20' and '30' which are moderate values residing in the limits. At first sight, we observe that when the dependencies between data types and applications increase, it becomes problematic to store the data applications need. Thus, **LPAC** can barely place its data in the FCUs. The required total capacity for the application-centric approach rises, and using this method penalizes the service providers from storage cost perspective. Required total capacity is again unaffected from the increases in FCU count and excess use in data-centric approaches as in the case of **appUseRatioInFCU**.

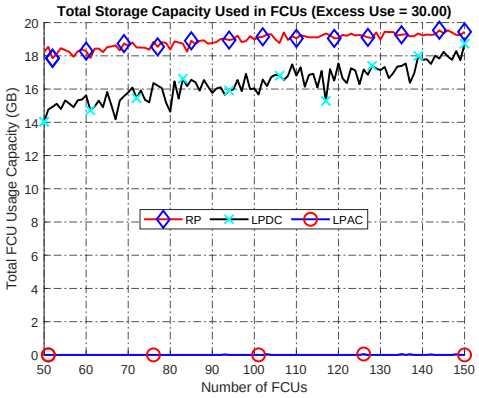
There exists again and inverse proportion between the chosen parameter and



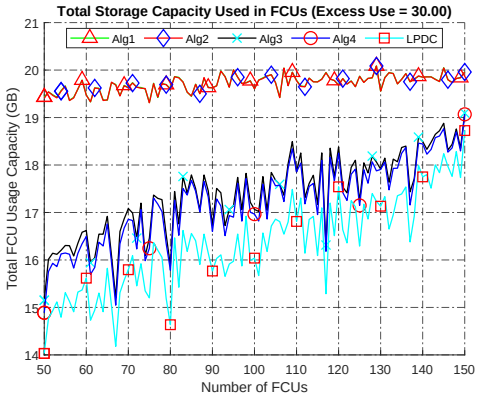
(a) Used CC Capacity in LPAC, LPDC and RP.



(b) Used CC Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.



(c) Used FCU Capacity in LPAC, LPDC and RP.



(d) Used FCU Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.

Figure 7.25: Total used CC & FCU storage capacities (appUseRatioInFCU = 0.05, EU = 30.0).

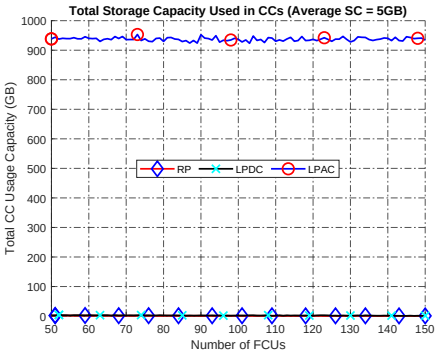
the required storage capacity in the FCUs. It is inevitable since the dependencies between the applications and data types rise, and although the number of running applications does not change, their requirements to different data types increase as this parameter goes up. This forces the data-centric approaches to choose places in between the FCUs where the latency cost is decreased.

If we consider the algorithms individually, their behavior look similar in the **appUseRatioInFCU** case. First two algorithms almost place all their data to FCUs, and for the optimal solution together with the remaining two algorithms, there is a balance between CCs and FCUs. From the total storage perspective,

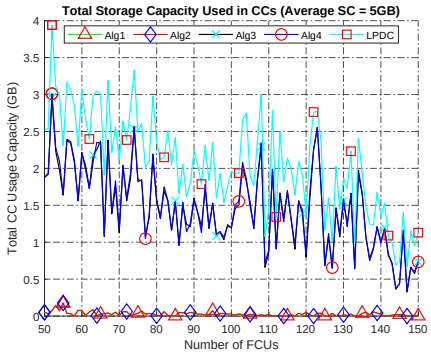
when the FCU count increases, total capacity used in the FCUs also rises as in the previous case, and all the data-centric approaches need only '20GB' of total storage. On the contrary of this, the required capacity of the application-centric approach is more than '920GB' in the worst case.

If we sum up the results regarding excess use, it is much more beneficial to use the data-centric approaches instead of traditional application-centric one in a highly data and application dependent network topology.

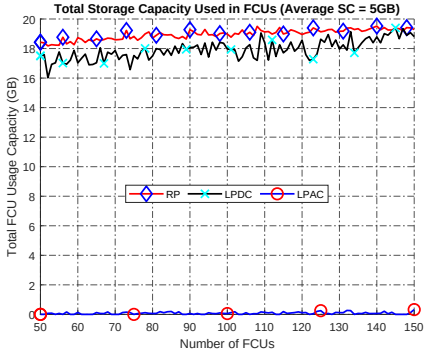
7.4.3 Effect of Storage Capacities on Data Storage



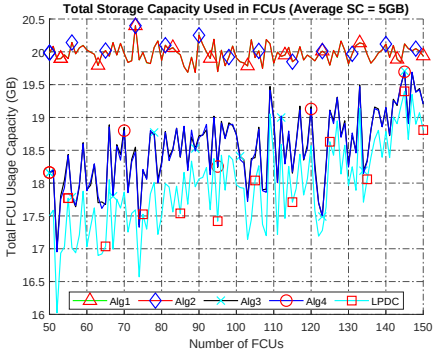
(a) Used CC Capacity in LPAC, LPDC and RP.



(b) Used CC Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.

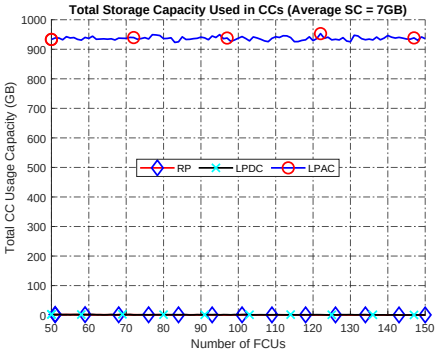


(c) Used FCU Capacity in LPAC, LPDC and RP.

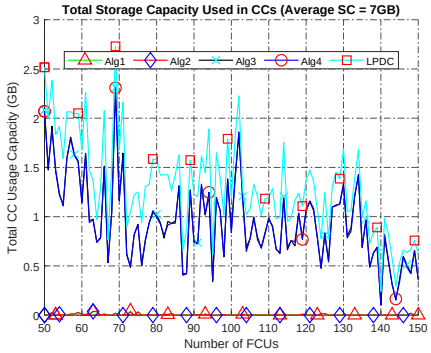


(d) Used FCU Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.

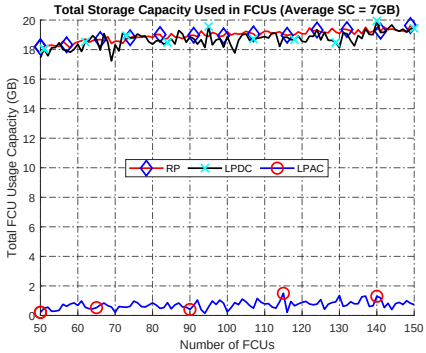
Figure 7.26: Total used CC & FCU storage capacities (Average capacity in an FCU = 5GB).



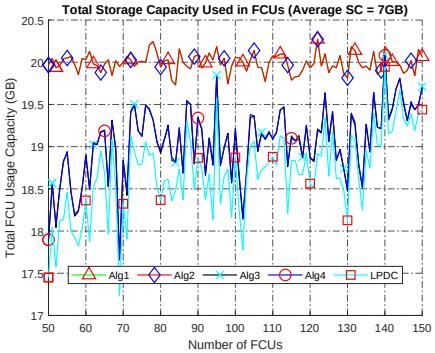
(a) Used CC Capacity in LPAC, LPDC and RP.



(b) Used CC Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.



(c) Used FCU Capacity in LPAC, LPDC and RP.

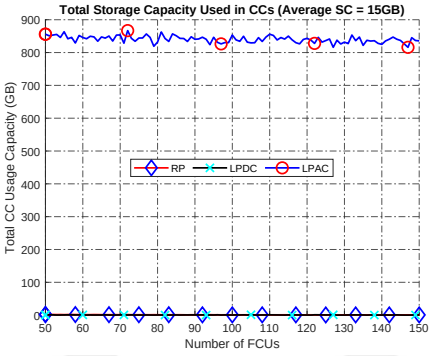


(d) Used FCU Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.

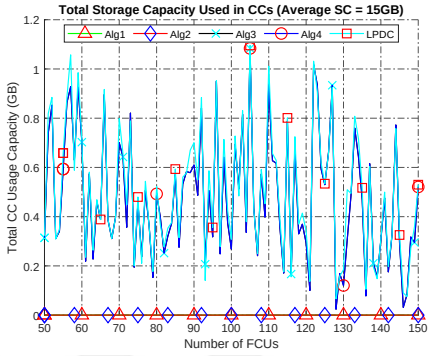
Figure 7.27: Total used CC & FCU storage capacities (Average capacity in an FCU = 7GB).

As a final scenario of the storage discussion, we change the available capacities in the FCUs and observe the results. We fix application run ratio and excess use parameters to '0.1' and '30', respectively. We sweep the means of the available storage resources in the FCUs from '5' to '15' together with the number of FCUs exist in the network model (see Figures 7.26 to 7.28).

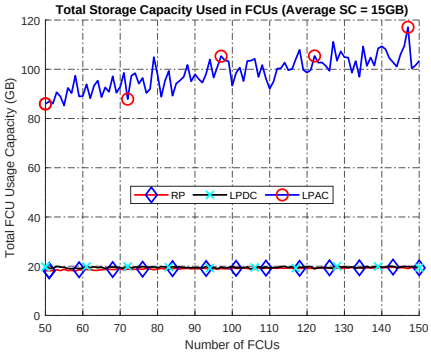
In these group of simulations, we try to see how FCUs are preferable if their capacities are high enough to store multiple different types of data. According to the results, it is easily seen that if the storage capacities of FCUs increase, all algorithms try to place their data to them. It is inevitable since we use mesh logical topology and some of the FCUs reside almost in equal distances to the others, these FCUs are chosen as the best places for the data types. When the



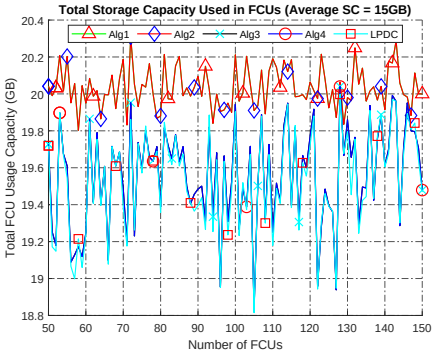
(a) Used CC Capacity in LPAC, LPDC and RP.



(b) Used CC Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.



(c) Used FCU Capacity in LPAC, LPDC and RP.



(d) Used FCU Capacity in Alg1, Alg2, Alg3, Alg4 and LPDC.

Figure 7.28: Total used CC & FCU storage capacities (Average capacity in an FCU = 15GB).

capacity of these FCUs are limited, some of the data types should be placed in sub-optimal places, and this increases the average latency applications encounter while reaching their required data types. This is also a validation of what we try to make with the data-centric approach: in limited capacities we should choose better places such that the average latency applications encounter is minimized, and these places are FCUs mostly. So while the storage capacities of the FCUs grow indefinitely, much more data can be placed in them and the average latency becomes minimum.

In this section we present the results obtained from the storage capacity perspective, and it is obvious that using the data-centric approaches in a network

where highly data dependent applications run, dramatically decreases the required data storage costs. Also as a conclusion, reason of the rise in average latency when **appUseRatioInFCU** and excess use parameters increase is not the storage capacities. It mostly depends on the relationship between applications and their required data types.

7.5 Algorithm Run-Time Results

We want to replace the linear model with heuristic algorithms, and make them possible for using in dynamic environments. All the proposed algorithms can run in any of the available cloud data centers in the network when a drastic change is observed in the data generation and access patterns or whenever new placement is wanted. Hence, the run-times of the algorithms become important, and in this section we want to investigate them.

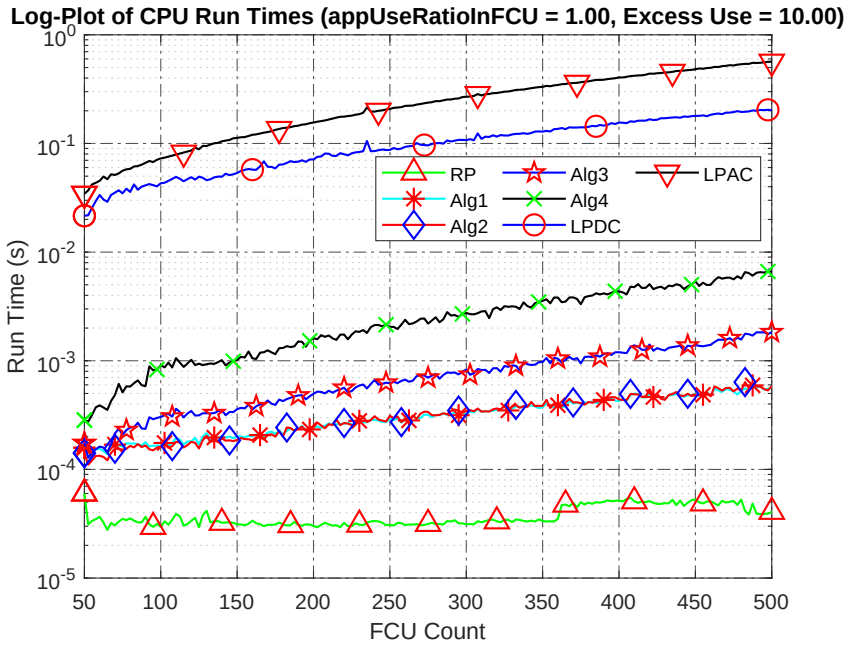


Figure 7.29: CPU run times of algorithms (appUseRatioInFCU = 1.00, EU = 10.0).

Run-time analyses of the algorithms according to model parameters are given

in Chapter 5. In this section, we want to compare these run-times with the run-times of the linear solvers. We come up with almost the same graph in all simulations we run so far, hence we choose just one of them to be presented here (Figure 7.29). To see the effect of the parameters clearly, we use logarithmic plot instead of a linear one.

As expected while the number of FCUs (**N**) increases, run-times of all algorithms increase. Obtaining the optimal solutions from linear solvers become time-consuming, when the total number of FCUs in the network is large, and this makes them difficult to adapt in a highly dynamic network.

When we consider the run-times of our proposed algorithms, increase in **Alg4** is more than the others. As mentioned in Section 7.3, although **Alg4** gives the best results among the proposed algorithms, its run-time is worse than the others; but it stills much better than the linear solvers. **Alg1** and **Alg2** require almost the same CPU run-time which verifies analyses made in Section 5.1 & 5.2.

To sum up, all of out proposed data-centric algorithms outperform from CPU run-time point of view the linear solvers, and they are adequate to be used in a highly dynamic and crowded environments to decrease the storage and latency costs.

7.6 Network Occupancy Results

Until now, we consider whether better average latency with less storage goal is achieved or not. Now, we want to be sure that we do not inject extra traffic to the network while reaching our goal, since it is not preferable. In order to calculate the network occupancy, we mainly focus on the traffic demand of the FCUs. As applications try to reach their required data types according to the access frequencies (**AF**) and run ratios (**AR**), they generate a demand for the corresponding data types. When we multiply these matrices together with total data generation volume of the data type, we can find traffic demands of the

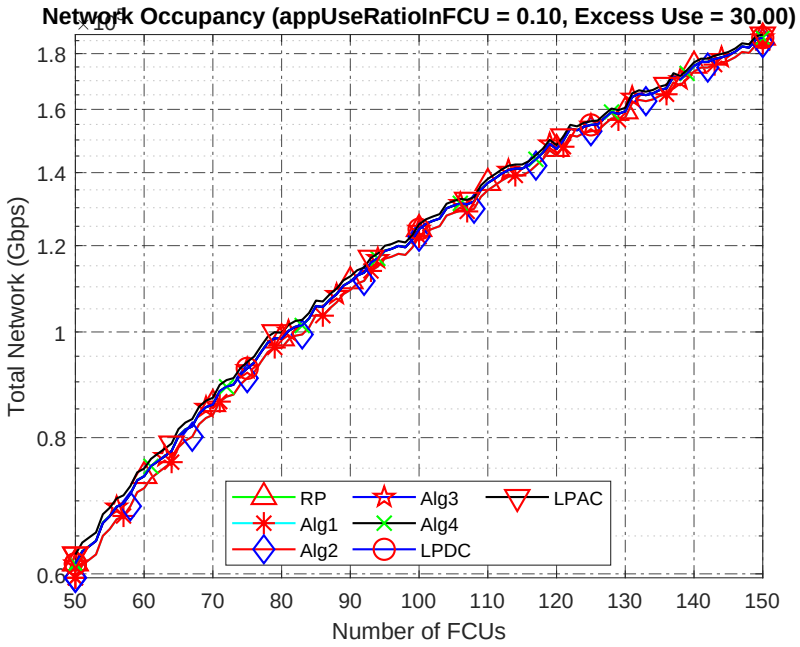


Figure 7.30: Network occupancy (appUseRatioInFCU = 0.1, EU = 30).

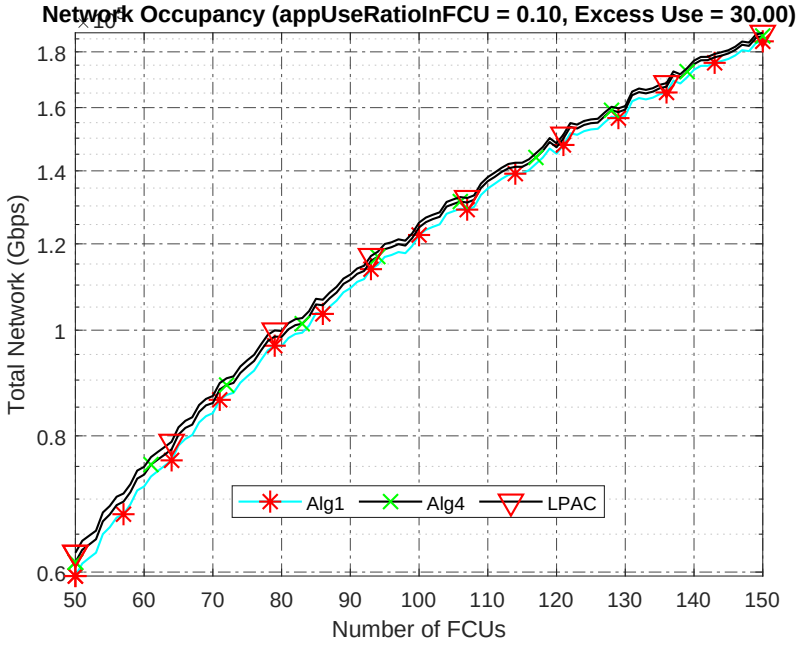


Figure 7.31: Network occupancy (appUseRatioInFCU = 0.1, EU = 30).

applications running. This is the baseline for calculating the network occupancy.

We examine all the results of the simulations, and the result we obtain is the same: there is no change from network occupancy perspective, and the application-centric approach is worse than the data-centric ones. Since all the results are similar from network occupancy perspective, we use the following two figures (Figure 7.30 & 7.31) showing this.

In the first figure (Figure 7.30), all algorithms are plotted and in the second figure (Figure 7.31), only **LPAC**, **Alg1** and **Alg4** are plotted. It is easily seen that there is a direct relation between the total number of FCUs and the network occupancy. It is inevitable that when the total number of FCUs increases, demand also increases; but the interesting thing is in all simulations **LPAC** gives worse results than the data-centric placement algorithms, and it concludes that preferring the data-centric placement instead of the application-centric one also decreases network occupancy slightly.

If data-centric algorithms are considered individually, **Alg1** & **Alg2** give slightly better results than other data-centric approaches. So we can conclude that placing mostly generated data types near to where they are used most decreases network occupancy slightly.

7.7 Summary

In this chapter, we present our simulation results to evaluate the performance of the linear models of both application-centric and data-centric approaches together with our proposed heuristic algorithms.

The results verify that the application-centric approach has some performance drawbacks from the required storage capacity perspective where there exists lots of different applications, and it has worse performance from latency perspective when the network gets crowded and the data dependencies of the applications increase. In these cases, the data-centric approaches outperform significantly.

Among the data-centric approaches, we observe that **Alg1** and **Alg2** give similar results in the simulations, and we conclude that in chosen scenarios placing mostly generated (**Alg1**) and mostly accessed (**Alg2**) data types near where they are required mostly give the same results; but from the network occupancy point of view they have a little advantage.

The optimal latency results are obtained by solving the linear model defined in Section 4.1 with linear solvers such as Gurobi [68]. The optimal result requires a balance between CCs and FCUs to place data. Although the best results are obtained from linear solvers, there is a significant drawback from run-time perspective, as the number of nodes in the network increases, it takes long time to find the optimal solution. At this point our proposed last two algorithms come into play. **Alg3** and **Alg4** are very good alternatives of the linear model, and they give only '0.1%' worse results when the average latency is considered. Their run-times are better than the linear model even if the number of nodes in the network increases. **Alg3** has better performance from run-time point of view, but **Alg4** is good at the latency.

To sum up, using the data-centric approaches instead of the application-centric gives significant advantage to the service providers, and our proposed **Alg3** and **Alg4** algorithms give almost the same results as the optimal solutions.

Chapter 8

Conclusion and Future Work

In this thesis, we first propose a cloud computing (CC) and fog computing (FC) based IoT network model for efficiently placing and serving IoT big data. We consider the fact that same type of IoT data may be needed and used by multiple applications. Therefore, we propose classification of the IoT data into types, and identification of which applications may require which types of data (Chapter 3). On top of that model, we define the data placement problem as an optimization model (Chapter 4) where the average latency encountered by applications in accessing their required data has to be minimized.

Proposed data placement strategy depends on the classification of data into types for reducing the network storage costs using the Fog Computing Units (FCUs). Data Classifier (DC) and Data Profiler (DP) agents run in fog computing data centers are responsible for the classification of the 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 name the data-centric placement, and it significantly reduces required storage costs for handling the big data generated in IoT networks.

Finding the optimal solution from latency perspective gets time consuming when the crowded networks proliferate, and for getting over this problem we

propose four heuristic algorithms (Chapter 5). All of these algorithms depend on the classification of data into types as in the linear case, and they become very effective when the number of applications, data types, node counts increase.

Since there is no known deployed network model consisting of the elements we use, we need to model the network for the performance evaluation of both linear models and the proposed algorithms. We model a rectangular network assuming it as a smart city with smaller rectangles, which are neighborhoods, and present the algorithms used during generation of the model (Chapter 6).

We conduct extensive simulations using the network model generated to evaluate all proposed heuristic algorithms together with the linear solutions, and present them in Chapter 7. Our simulation results show that our algorithms can efficiently place data without increasing average latency that applications encounter. We also observe that when application running frequencies on FCUs and excess use between applications and data types increase, the data-centric placement strategies perform much better than the application-centric data placement where each application stores their 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 heuristic algorithms with meta-heuristic algorithms, such as genetic algorithms, simulated annealing, particle swarm optimization (PSO), as a future work. We also leave the details of establishing the 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 data placement in this thesis, they may be required for some cases, for instance replication becomes important when the reliability is critical or when the network satisfy the QoS needs of a delay critic application. These issues are left as potential future research areas of our work presented in this thesis.

Our work is complementary with some other approaches and methods that try to optimize the 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 thesis will guide fog-cloud 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 proliferate.

Bibliography

[1] "ITU Internet Reports The Internet of Things," tech. rep., ITU, Geneva, Nov 2005.

[2] J. E. Ibarra-Esquer, F. F. Gonzalez-Navarro, B. L. Flores-Rios, L. Burtseva, and M. A. Astorga-Vargas, "Tracking the evolution of the internet of things concept across different application domains," *Sensors*, vol. 17, no. 6, 2017.

[3] D. Evans, "The Internet of Things How the Next Evolution of the Internet is Changing Everything." `https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf`, Apr 2011. (accessed 28 July 2018).

[4] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010.

[5] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[6] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645 – 1660, 2013.

[7] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer, and P. Doody, "Internet of Things Strategic Research Roadmap," in *Internet of Things - Global*

Technological and Societal Trends From Smart Environments and Spaces to Green ICT (O. Vermesan and P. Friess, eds.), ch. 2, pp. 9–52, Aalborg: River Publishers, 2011.

[8] J. A. Stankovic, "Research directions for the internet of things," *IEEE Internet of Things Journal*, vol. 1, pp. 3–9, Feb 2014.

[9] S. Hiremath, G. Yang, and K. Mankodiya, "Wearable internet of things: Concept, architectural components and promises for person-centered healthcare," in *2014 4th International Conference on Wireless Mobile Communication and Healthcare - Transforming Healthcare Through Innovations in Mobile and Wireless Technologies (MOBIHEALTH)*, pp. 304–307, Nov 2014.

[10] T. Nam and T. A. Pardo, "Conceptualizing smart city with dimensions of technology, people, and institutions," in *Proceedings of the 12th Annual International Digital Government Research Conference: Digital Government Innovation in Challenging Times*, dg.o '11, (New York, NY, USA), pp. 282–291, ACM, 2011.

[11] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, "An information framework for creating a smart city through internet of things," *IEEE Internet of Things Journal*, vol. 1, pp. 112–121, April 2014.

[12] G. Suciu, A. Vulpe, S. Halunga, O. Fratu, G. Todoran, and V. Suciu, "Smart cities built on resilient cloud computing and secure internet of things," in *2013 19th International Conference on Control Systems and Computer Science*, pp. 513–518, May 2013.

[13] W. Ejaz, M. Naeem, A. Shahid, A. Anpalagan, and M. Jo, "Efficient energy management for the internet of things in smart cities," *IEEE Communications Magazine*, vol. 55, pp. 84–91, January 2017.

[14] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of lpwan technologies for large-scale iot deployment," *ICT Express*, vol. 5, no. 1, pp. 1 – 7, 2019.

[15] A. Ikpehai, B. Adebisi, K. M. Rabie, K. Anoh, R. E. Ande, M. Hammoudeh, H. Gacanin, and U. M. Mbanaso, "Low-power wide area network technologies for internet-of-things: A comparative review," *IEEE Internet of Things Journal*, vol. 6, pp. 2225–2240, April 2019.

[16] M. R. Palattella, M. Dohler, A. Grieco, G. Rizzo, J. Torsner, T. Engel, and L. Ladid, "Internet of things in the 5g era: Enablers, architecture, and business models," *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 510–527, March 2016.

[17] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, pp. 32–39, Nov 2011.

[18] Y. Jie, J. Y. Pei, L. Jun, G. Yun, and X. Wei, "Smart home system based on iot technologies," in *2013 International Conference on Computational and Information Sciences*, pp. 1789–1791, June 2013.

[19] M. Yun and B. Yuxin, "Research on the architecture and key technology of internet of things (iot) applied on smart grid," in *2010 International Conference on Advances in Energy Engineering*, pp. 69–72, June 2010.

[20] P. Scully, "The Top 10 IoT Segments in 2018 - based on 1600 real IoT projects." `http://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/`, Feb 2018. (accessed 28 July 2018).

[21] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, pp. 22–32, Feb 2014.

[22] H. Schaffers, N. Komninos, M. Pallot, B. Trousse, M. Nilsson, and A. Oliveira, "Smart Cities and the Future Internet: Towards Cooperation Frameworks for Open Innovation," in *The Future Internet*, (Berlin), pp. 431–446, Springer, 2011.

[23] L. D. Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 10, pp. 2233–2243, Nov 2014.

[24] E. Ahmed, I. Yaqoob, I. A. T. Hashem, I. Khan, A. I. A. Ahmed, M. Imran, and A. V. Vasilakos, "The role of big data analytics in internet of things," *Computer Networks*, vol. 129, pp. 459 – 471, 2017. Special Issue on 5G Wireless Networks for IoT and Body Sensors.

[25] M. Chen, S. Mao, and Y. Liu, "Big Data: A Survey," *Mobile Networks and Applications*, vol. 19, pp. 171–209, Apr 2014.

[26] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog Computing: A platform for Internet of Things and Analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments* (N. Bessis and C. Dobre, eds.), pp. 169–186, Cham: Springer, 2014.

[27] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," tech. rep., University of California at Berkeley, Feb 2009.

[28] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of "big data" on cloud computing: Review and open research issues," *Information Systems*, vol. 47, pp. 98 – 115, 2015.

[29] A. Botta, W. de Donato, V. Persico, and A. Pescape, "Integration of Cloud computing and Internet of Things: A survey," *Future Generation Computer Systems*, vol. 56, pp. 684 – 700, 2016.

[30] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, (New York, NY, USA), pp. 13–16, ACM, 2012.

[31] S. Yi, C. Li, and Q. Li, "A Survey of Fog Computing: Concepts, Applications and Issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, (New York, NY, USA), pp. 37–42, ACM, 2015.

[32] I. Stojmenovic and S. Wen, "The Fog Computing Paradigm: Scenarios and Security Issues," in *2014 Federated Conference on Computer Science and Information Systems*, vol. 2, pp. 1–8, 2014.

[33] L. M. Vaquero and L. Rodero-Merino, "Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *SIGCOMM Computer Communication Review*, vol. 44, pp. 27–32, Oct 2014.

[34] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog computing: Survey of trends, architectures, requirements, and research directions," *IEEE Access*, vol. 6, pp. 47980–48009, 2018.

[35] M. Ashouri, P. Davidsson, and R. Spalazzese, "Cloud, edge, or both? towards decision support for designing iot applications," in *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, pp. 155–162, Oct 2018.

[36] N. K. Giang, M. Blackstock, R. Lea, and V. C. M. 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, Oct 2015.

[37] J. Deichmann, K. Heineke, T. Reinbacher, and D. Wee, "Creating a successful Internet of Things data marketplace." `http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/creating-a-successful-internet-of-things-data-marketplace`, Oct 2016. (accessed 28 July 2018).

[38] F. Karatas and I. Korpeoglu, "Fog-Based Data Distribution Service (F-DAD) for Internet of Things (IoT) applications," *Future Generation Computer Systems*, vol. 93, pp. 156 – 169, Apr 2019.

[39] J. Santos, T. Vanhove, M. Sebrechts, T. Dupont, W. Kerckhove, B. Braem, G. V. Seghbroeck, T. Wauters, P. Leroux, S. Latre, B. Volckaert, and F. D. Turck, "City of things: Enabling resource provisioning in smart cities," *IEEE Communications Magazine*, vol. 56, pp. 177–183, July 2018.

[40] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Fog Computing: Enabling the Management and Orchestration of Smart City Applications in 5G Networks," *Entropy*, vol. 20, pp. Art. Num. 4, 1–26, Jan 2018.

[41] M. Aazam and 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*, pp. 464–470, Aug 2014.

[42] M. Jutila, "An Adaptive Edge Router Enabling Internet of Things," *IEEE Internet of Things Journal*, vol. 3, pp. 1061–1069, Dec 2016.

[43] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755 – 768, 2012.

[44] Z. Guo, S. Hui, Y. Xu, and H. J. Chao, "Dynamic Flow Scheduling for Power-Efficient Data Center Networks," in *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, pp. 1–10, June 2016.

[45] Z. Guo, Z. Duan, Y. Xu, and H. J. Chao, "Cutting the Electricity Cost of Distributed Datacenters Through Smart Workload Dispatching," *IEEE Communications Letters*, vol. 17, pp. 2384–2387, December 2013.

[46] Z. Guo, Z. Duan, Y. Xu, and H. J. Chao, "JET: Electricity cost-aware dynamic workload management in geographically distributed datacenters," *Computer Communications*, vol. 50, pp. 162 – 174, 2014.

[47] V. Angelakis, I. Avgouleas, N. Pappas, E. Fitzgerald, and D. Yuan, "Allocation of Heterogeneous Resources of an IoT Device to Flexible Services," *IEEE Internet of Things Journal*, vol. 3, pp. 691–700, Oct 2016.

[48] C.-W. Tsai, "SEIRA: An effective algorithm for IoT resource allocation problem," *Computer Communications*, vol. 119, pp. 156 – 166, 2018.

[49] I. Lera, C. Guerrero, and C. Juiz, "Comparing centrality indices for network usage optimization of data placement policies in fog devices," in *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 115–122, April 2018.

[50] 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, May 2017.

[51] Z. Rezazadeh, D. Rahbari, and M. Nickray, "Optimized module placement in iot applications based on fog computing," in *Electrical Engineering (ICEE), Iranian Conference on*, pp. 1553–1558, May 2018.

[52] Z. Rezazadeh, M. Rezaei, and M. Nickray, "Lamp: A hybrid fog-cloud latency-aware module placement algorithm for iot applications," in *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*, pp. 845–850, Feb 2019.

[53] N. B.V. and R. M. R. Guddeti, "Heuristic-based iot application modules placement in the fog-cloud computing environment," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pp. 24–25, Dec 2018.

[54] K. Toczé and S. Nadjm-Tehrani, "A Taxonomy for Management and Optimization of Multiple Resources in Edge Computing," *Wireless Communications and Mobile Computing*, vol. 2018, pp. Art. ID 7476201, 1–23, 2018.

[55] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption," *IEEE Internet of Things Journal*, vol. 3, pp. 1171–1181, Dec 2016.

[56] L. Tong, Y. Li, and W. Gao, "A Hierarchical Edge Cloud Architecture for Mobile Computing," in *IEEE INFOCOM 2016 - IEEE International Conference on Computer Communications*, pp. 1–9, April 2016.

[57] 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, pp. 3702–3712, Dec 2016.

[58] H. R. Arkian, A. Diyanat, and A. Pourkhalili, "MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications," *Journal of Network and Computer Applications*, vol. 82, pp. 152 – 165, 2017.

[59] 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, Nov 2016.

[60] R. Yu, G. Xue, and X. Zhang, "Application Provisioning in Fog Computing-enabled Internet-of-Things: A Network Perspective," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 783–791, April 2018.

[61] Y. Qin, Q. Z. Sheng, N. J. Falkner, S. Dustdar, H. Wang, and A. V. Vasilakos, "When things matter: A survey on data-centric internet of things," *Journal of Network and Computer Applications*, vol. 64, pp. 137 – 153, 2016.

[62] B. Yu and J. Pan, "Location-aware Associated Data Placement for Geo-distributed Data-intensive Applications," in *IEEE INFOCOM 2015 - 2015 IEEE Conference on Computer Communications*, pp. 603–611, April 2015.

[63] B. Tang, Z. Chen, G. Hefferman, T. Wei, H. He, and Q. Yang, "A Hierarchical Distributed Fog Computing Architecture for Big Data Analysis in Smart Cities," in *Proceedings of the ASE BigData & SocialInformatics 2015*, (New York, NY, USA), pp. Art. Num. 28, 1–6, ACM, Oct 2015.

[64] S. M. A. Oteafy and H. S. Hassanein, "IoT in the Fog: A Roadmap for Data-Centric IoT Development," *IEEE Communications Magazine*, vol. 56, pp. 157–163, Mar 2018.

[65] A. Aliyu, A. H. Abdullah, O. Kaiwartya, Y. Cao, J. Lloret, N. Aslam, and U. M. Joda, "Towards video streaming in IoT Environments: Vehicular communication perspective," *Computer Communications*, vol. 118, pp. 93 – 119, Mar 2018.

[66] E. Badidi, H. Routaib, and M. El Koutbi, "Towards data-as-a-service provisioning with high-quality data," in *Advances in Ubiquitous Networking 2* (R. El-Azouzi, D. S. Menasche, E. Sabir, F. De Pellegrini, and M. Benjillali, eds.), (Singapore), pp. 611–623, Springer, Nov 2017.

[67] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog Computing: Principles, Architectures, and Applications," in *Internet of Things* (R. Buyya and A. V. Dastjerdi, eds.), ch. 4, pp. 61 – 75, Morgan Kaufmann, 2016.

[68] G. Optimization, "Gurobi optimizer." `https://www.gurobi.com/products/gurobi-optimizer`, 2018. (accessed 31 May 2018).

[69] IBM, "Cplex optimizer." `http://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer`, 2018. (accessed 31 May 2018).

[70] V. V. Petrov, *Sums of Independent Random Variables*. Springer Verlag, 1975.

[71] MathWorks, "Matlab." `http://www.mathworks.com/products/matlab.html`, 2018. (accessed 05 June 2018).

[72] L. Atzori, A. Iera, G. Morabito, and M. Nitti, "The Social Internet of Things (SIoT) - When social networks meet the Internet of Things: Concept, architecture and network characterization," *Computer Networks*, vol. 56, pp. 3594 – 3608, Nov 2012.